BEST AVAILABLE COPY

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
19 April 2001 (19.04.2001)

PCT

(10) International Publication Number
WO 01/28060 A1

(51) International Patent Classification[7]: H02H 3/05

(21) International Application Number: PCT/IL00/00639

(22) International Filing Date: 10 October 2000 (10.10.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
132328 11 October 1999 (11.10.1999) IL

(71) Applicant *(for all designated States except US)*: COM-LOG TELECOMMUNICATIONS ENGINEERING LTD. [IL/IL]; Rehov HaTaas 23, 44425 Kfar Saba (IL).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: EFRATI, Ofir [IL/IL]; 39 Pardes Mishutaf St., 43355 Raanana (IL).

(74) Agent: LANGER, Edward; P.O. Box 410, 43103 Raanana (IL).

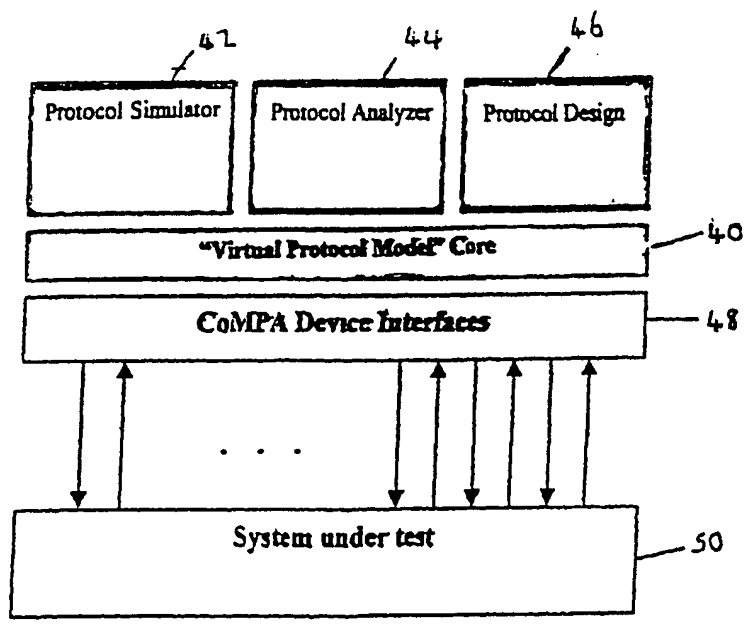
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*
— *Before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: DIGITAL TESTING DEVICE



(57) Abstract: A digital testing device using protocols to evaluate a system under test (50).

WO 01/28060 A1

Digital Testing Device

Field of the Invention

The present invention relates to a digital testing device, and more particularly but not exclusively to a digital testing device and method for use in the data communications and telecommunications fields.

Background of the Invention

Modern telecommunications systems are generally constructed of various elements capable of data processing and data communication support. The various elements may be interlinked in order to establish reliable data flow between the elements and in order to guarantee proper harmonic operation as one complete system. In passing through such systems digital data can be put through numerous complex processes, each using numerous protocols. For example, in digital communications, switching, modulating, encoding, decoding and numerous other operations are needed. Each operation involves a different algorithm or protocol or series thereof that must be thoroughly tested. The protocols may be standard protocols or they may be proprietary protocols unique to the equipment manufacturer. Frequently the manufacturer uses an in-house modification of a standard protocol. In general, each different protocol or group of similar protocols requires a different test device, and often even minor variations of the algorithm may require the use of a different device. A single communication system may combine equipment of several manufacturers. It may thus require numerous test devices and be very difficult to test. Furthermore, the different test devices generally do not work together and thus test of integrated scenarios is generally not possible.

Manufacturers find the use of proprietary algorithms helpful, however, one of the disadvantages of using such an algorithm is that time to market is delayed whilst a suitable test device can be designed and perfected.

In addition, data links occasionally make use of security algorithms such as encryption systems, which need to be safeguarded, and some manufacturers may wish to keep their proprietary algorithms secret.

Summary Of The Invention

In accordance with a first aspect of the present invention there is provided a digital testing device comprising: at least one data interface, a plurality of predetermined protocols, a protocol selector for selecting any combination of said protocols to operate on said digital data, and an output for outputting results of applying said digital data to said selected protocols.

According to a second aspect of the present invention there is provided a digital testing device comprising: at least one data interface, a user interface, a protocol constructor operable to accept data from said user interface and to construct a protocol in accordance therewith, said protocol being able to operate on said digital data, and an output for outputting results of applying said digital data to said selected protocols.

According to a third aspect of the present invention there is provided a digital testing device comprising, at least one data interface, a plurality of predetermined protocols, a user interface, a protocol constructor operable to accept data from said user interface and to construct an additional protocol in accordance therewith, said protocol being able to operate on said digital data, a protocol selector for selecting any combination of said protocols to operate on said digital data and an output for outputting results of applying said digital data to said selected protocols..

A fourth aspect of the present invention comprises a digital simulation device which comprises the protocols, a protocol constructor, the user interface, a protocol selector and the output. Instead data is produced automatically within the device itself and sent via the interface

Preferably, the digital data is the output of all or part of a communications device. The device may be comprised within a computer. The output may comprise a configurable data protocol which may be set to find points of interest in the data under test. The input may comprise a selectable one of a plurality of device interfaces. The device interfaces are preferably designed with specific digital equipment in mind and may be operable to hide characteristics of the equipment from the device. Means are provided for enabling the user to generate new device interfaces or to configure existing device interfaces.

Embodiments of the invention are operable with any digital data.

The protocols referred to above preferably synthesize the data format of various communication protocols as used in communications engineering.

The protocols are preferably descendant objects of a protocol-independent parent object, which may alternatively be referred to as a virtual protocol. The protocol-independent parent object is preferably operable to support processing of data according to any predefined protocol, which may be a descendant object thereof. The algorithm-independent parent object may

support certain desirable features such as smart filtering, triggering and simulation operations, which enhance testing ability. The protocols may be templates for arranging incoming data into the data field and packet format of any predetermined protocol or other communication algorithm.

The protocol constructor preferably comprises a bank of predefined protocol substeps represented by visual items in the user interface, individually selectable by a user through said user interface to build a protocol.

The specific device interfaces referred to above are preferably represented by visual items in the user interface, individually selectable by a user through said user interface.

Embodiments of the invention may thus provide a multi-interface, multi-protocol analyzer and simulator for testing or simulating complex communication systems involving any standard or proprietary protocol. The embodiments are based on a virtual protocol model.

In an embodiment, automatic association may be made between a given device interface and one of said plurality of protocols. In a further embodiment, external test devices may be integrated into the system using a suitably designed interface.

Embodiments of the invention may provide a device that can test an entire telecommunication system comprising equipment from numerous manufacturers, using a plurality of ports each of which can be programmed to operate on a different protocol, and which can be associated logically one with the other. The device may enable a reduction in development time and cost.

Brief Description of the Drawings

For a better understanding of the invention and to show how the same may be carried into effect, reference is now made, purely by way of example, to the accompanying drawings, in which:

Fig. 1 is a simplified diagram of a first embodiment of a device according to the invention,

Fig. 2 is a generalized block diagram of a device according to a preferred embodiment of the present invention,

Fig. 3 is a generalized layer diagram of an embodiment of the present invention,

Fig. 4 is a generalized diagram showing in more detail the virtual protocol model core of Fig. 3,

Fig. 5 is a screen view of an exemplary protocol of the kind described in Fig. 4, as seen from within the protocol designer,

Fig. 6 shows the first step in the process of designing an object for testing a proprietary protocol,

Fig. 7 is a screen view showing how the user interface 26 permits selection of a prestored interface for the interface object,

Fig. 8 is a screen view of available protocols including that defined in Fig. 6,

Fig. 9 is a screen view showing a new logical channel configured with a protocol and physical interface,

Fig. 10 is a tree diagram showing a tree object based on the protocol shown in Fig. 6,

Fig. 11 is a screen view showing the monitor output when test data bytes arrive and fit into the protocol tree as shown in Fig. 10, and

Fig. 12 is a screen view showing a capture filter for monitoring the results of the analysis of Fig. 11.

Description of the Preferred Embodiments

Reference is firstly made to Fig. 1, which is a simplified diagram of a first embodiment of a device according to the invention. An interface connector 10 is attachable to a telecommunications system unit 12 or other communication device, in such a way that it is able to extract data for testing. The interface connector is attached to a portable, or other, computer 14, which carries out an analysis of the extracted data. The interface connector is simply a buffering device allowing data from the exchange unit 12 to reach an input port of the computer 14. In some embodiments, buffering may not be necessary.

The use of a general-purpose computer for carrying out the analysis of the extracted data is preferred but not essential. As an alternative, it is possible to use a dedicated digital device.

Reference is now made to Fig. 2, which is a generalized block diagram of a device according to a preferred embodiment of the present invention. It is to be borne in mind that the invention does not solely encompass the device configured for use but also relates to the features of the device that aid easy configuration and also to the method of configuration.

In Fig. 2, a device interface 20 provides interfacing with the equipment to be tested. The interface is preferably selected to be specific to the equipment and is operable to hide hardware features of the equipment under test from the device.

A protocol bank 22 comprises a plurality of predefined protocols $P_1 \ldots P_N$, all of which are designed to simulate the operation, on a datastream, of a specific item of equipment. The relevant protocol or protocols are selected by the user through a protocol selector screen of a

user interface (Fig. 8 below). The user interface is preferably a visual interface; wherein objects to be selected by the user are generally presented as graphical icons.

In the event that the desired protocol is not present, the device allows for the easy addition of new protocols. This may be done by obtaining a predefined protocol from an external source such as the Internet, or it may be done by use of a protocol designer (46 in Fig.3). The protocol designer 46 contains a bank of protocol substeps (not shown), which are preferably represented by icons in the user interface. The user is able to select from the substeps to build his own protocol P_{New}, as will be explained in greater detail below. The newly formed protocol is then available for protocol selection in the same way as the predefined protocols $P_1...P_N$.

An output unit 32, again configurable through the user interface, filters the output data for interpretation by the user. User configurable filters $F_1...F_M$ are stored in a filter bank 24 and customization thereof will be discussed below with respect to Fig. 12. User configurable triggers $T_1...T_K$, for triggering test operations and the like, are stored in a trigger bank 26 and are likewise customizable through the user interface.

As will be discussed below, devices according to the present invention may run simulations as well as tests. Thus a bank of prestored and configurable simulations is stored in a simulation bank 28.

A virtual protocol model core 30 is a parent object that encapsulates a generic description of a communication protocol. It is preferably able to take on the characteristics of any protocols and the like that are applied to it. This is because all of the protocols etc. that have been discussed with respect to Fig. 2 are preferably defined as descendant objects thereof, as will be discussed in more detail below. An advantage of such a parent descendant arrangement is that when a protocol is prepared and or applied by a user, it requires no compilation or pre-processing but rather can be used immediately.

Reference is now made to Fig. 3, which is a generalized layer diagram of an embodiment of the present invention. A virtual protocol model core 40, as discussed above, is a parent object that encapsulates a generic description of a communication protocol. Three types of descendant objects of the core 40 are defined as a protocol simulator 42, a protocol analyzer 44 and a protocol designer 46. The protocol designer allows for the definition of a new protocol using the user interface as discussed above. A dedicated language is preferably used in order to facilitate the application of the virtual protocol by which the new protocol is to be described. The dedicated language preferably permits the creation of any combination of protocol layers by applying the object-oriented concept (inheritance & encapsulation). A simple linkage

between protocol components can lead to the construction of a wide and complex protocol library, where each protocol is built from very simple protocol components.

The user interface preferably provides a visual protocol designer utility, as mentioned above, which allows the construction of protocol components without the user needing to be familiar with the modeling language.

The protocol analyzer 44 allows for analysis of any protocol added to the protocol model core, or any set or combination of protocols. The protocol simulator 42 allows for the simulation of any protocol, or set or combination thereof added to the model core 40.

A further parent object is the device interface object 48, which is a generic model of a device data interface. A specific interface object appropriate to the system under test 50 and which is a descendant of the generic interface object, is preferably selected as described above. The model core, as mentioned, is preferably hardware independent. It is preferably data frame oriented, which means that any data pattern can be analyzed or generated as long as it is bit oriented. The specific interface object 48 permits easy interfacing with external hardware. The generic interface object 48 preferably comprises several interface functions that hide hardware dependent parameters.

A plurality of interfaces may be used at the same time so that more than one data channel or device may be tested simultaneously. Preferably, the device is configurable so that logical connections are provided between different channels. Thus, the data connected over numerous different channels can be analyzed as a single logical unit. Likewise, in simulation, several devices or data channels may be simulated simultaneously and logical connections may be configured between them.

A series of logical connections may be configured and particular series of data may be tested to produce logical scenarios. This may be done as part of testing or as part of a simulation or as a combination of the two.

If testing of communications equipment is being carried out then the interface is used to obtain data from the equipment. It can also return data to the equipment, as part of a simulation and thus serve as a link in an operational connection. The interface may be a single channel or it may be multi-channel, allowing the testing of combined systems or testing of combined systems simultaneously.

If simulation is being carried out then the data interface may be dispensed with, and an embodiment built solely for simulation need not comprise a data interface.

Reference is now made to Fig. 4, which is a generalized diagram showing in more detail the virtual protocol model core of Fig. 3. As mentioned above, the virtual protocol model is a

non-specific protocol analyzer. It analyzes collected data according to a predefined protocol, that is to say one of the protocols referred to above. The model is based on the OSI's layered model and supports among other the following main features:

Multi-layer analysis,

Any field, any data format and any intelligent algorithm concerning data processing, and

Multi channel analysis/data generation in parallel.

A tree structure is an optimal structure for presentation of any protocol, algorithm or filter. In general, a predefined protocol is maintained as a tree object, that is to say a descendent of the generalized tree structure. Figure 4 is a generalized tree structure, designed for an embodiment of the present invention, in which the various nodes can be defined with properties of the protocol (nodes 60, 62 64 and 66) or they may lead to a whole new layer of nodes, (sub-node 68). A second layer is shown explicitly but the only limit on the number of layers that may be included is the available memory. That is to say the number of layers is practically unlimited. The existence of one or more sub-layers, each of which can be constructed with any degree of complexity, gives the embodiments of the present invention the degree of flexibility necessary to apply testing to a wide variety of devices and protocols.

Each node in the tree may indicate a field, a group of fields or a sub-protocol layer. A field is an abstract term that covers a data pattern, ranging from a single bit to an endless chain of bit-data. A branch in the tree represents a branch in the protocol. Such a branch may, for example, be represented by the command "separate two messages by a message type field". Numerous other functions and properties may be added to the tree to allow flexibility and automation of the data process/generation.

Examples of the kind of properties that may be comprised in nodes within the tree are:

Automatic check sum data fields,

Variable length of data field according to dynamic conditions (dependent on other received data),

Data masking and pattern comparison,

Idle line timeout, and

Data size trash hold.

The virtual protocol model core permits the integration of externally provided data processes that have been generated by users in the way described above. The internal working of the externally provided processes are preferably totally hidden from the model core itself, and this is achieved by utilizing an external DLL (Dynamic Link Library). Any node in the virtual protocol tree can be assigned to such a DLL. This feature is important, for example if the user

wishes to safeguard protocols involving security applications and the like. The data processes referred to include both the protocols and the data interfaces. Both may be provided as external DLLs with their internal data processing being hidden from the core.

Reference is now made to Fig. 5, which is a screen view of an exemplary protocol of the kind described in Fig. 4, as seen from within the protocol designer 46. Various nodes are shown which define the parts of a data packet for the well-known TCP/IP protocol. It features internal branching for a higher protocol layer, as well as TCP and UDP protocols which are encapsulated as sub-protocol nodes denoted as "TCP packets" and "UDP packets".

In the aforementioned embodiments there is thus provided a testing device that is multi-protocol, multi-channel and multi-interface.

Reference is now made to Fig. 6, which shows the first step in the process of designing an object for testing a proprietary protocol. As in Fig. 5, there is shown a screen view of a protocol as seen from within the protocol designer 46. The protocol requires a definition of the data fields of a data packet and a series of nodes define open flag, message format connect, message format disconnect, data and close flag data fields. The definitions preferably include a size and a format for data within the field.

Reference is now made to Fig. 7, which is a screen view showing how the user interface 26 permits selection of a prestored interface for the interface object 48, and to Fig. 8 which is a screen view of available protocols including that defined in Fig. 6. A 'channel configuration manager' window (not shown), appears or is invoked and the user is directed to create a new logical channel and then select the required physical interface and assign a protocol. The physical interface objects are all presented as icons on the screen and the user simply clicks on the desired icon. Then the protocol defined in Fig. 6 is selected from the list shown in Fig. 8, which is a selection screen showing all previously defined screens.

Reference is now made to Fig. 9, which is a screen view showing the new logical channel, here labeled simply "my proprietary channel" configured with the protocol and physical interface previously selected. The process can be extended to add more channels and protocols, and thus to construct a comprehensive configuration for analyzing and simulating multi-channel systems to any desired degree of complexity.

The model is now ready for analysis. The analysis object is preferably activated to execute real time data capture and analysis. Data bits from the external device are captured by the interface. From the interface they are forwarded to the protocol tree where they filter through, starting at the root node and branching to other nodes and protocol layers, as though the data

were being fed through a real device according to the protocol rules. The data as received is fitted into the data fields as defined in the protocol, ready for further analysis.

Reference is now made to Fig. 10 which is a tree diagram showing a tree object based on the protocol shown in Fig. 6. In this example there is only a single protocol layer and a single branch. Now assume the following data bytes are captured: 0x02, 0x03, 0x02, 0x01, 0x01, 0x03 (0x – indicates hexadecimal digit). Fig. 10 shows the data bytes fitted into the fields of the protocol as discussed with respect to Fig. 9.

Reference is now made to Fig. 11, which shows the monitor output when the data bytes listed as above arrive and fit into the protocol tree as shown in Fig. 10.

Reference is now made to Fig. 12, which is a screen view showing a capture filter for monitoring the results of the analysis of Fig. 11. It will be apparent that for large quantities of data, the monitor output as shown in Fig. 11 could be difficult to interpret. There is thus provided a capture filter which can be set to look out for certain key features of the incoming data that it is desired to study. In the screen view the only messages that are studied are those wherein the fields "message format: disconnect" and "length of field" contain 0x2 values. All else is ignored.

If, instead of analyzing a particular device it is desired instead to test the operation of a particular protocol then the procedure is the same as described above except that the simulation object is entered before beginning the test. In the testing object, data is obtained from the interface, however, in the simulation object, data is produced, either at random or in a predetermined manner. The remainder of the operation is identical.

In an embodiment, automatic association may be made between a given device interface and one of said plurality of protocols. In a further embodiment, external test devices may be integrated into the system using a suitably designed interface.

It is appreciated that various features of the invention which are, for clarity, described in the contexts of separate embodiments may also be provided in combination in a single embodiment. Conversely, various features of the invention which are, for brevity, described in the context of a single embodiment may also be provided separately or in any suitable subcombination.

It will be appreciated by persons skilled in the art that the present invention is not limited to what has been particularly shown and described hereinabove. Rather, the scope of the present invention includes both combinations and subcombinations of the various features described hereinabove as well as variations and modifications thereof which would occur to persons skilled in the art upon reading the foregoing description and which are not in the prior art.

The source code for the virtual protocol core analyzer is given in the following tables:

Claims

1. A digital testing device comprising:

at least one data interface for interfacing to at least one device to be tested,

a plurality of predetermined protocols,

a protocol selector for selecting any combination of said protocols to operate on digital data obtained via said interface, and

an output for outputting results of applying said digital data to said selected protocols.

2. A digital testing device comprising:

at least one data interface,

a user interface,

a protocol constructor operable to accept data from said user interface and to construct a protocol in accordance therewith, said protocol being able to operate on digital data obtained from said interface, and

an output for outputting results of applying said digital data to said selected protocols.

3. A digital testing device comprising:

at least one data interface,

a plurality of predetermined protocols,

a user interface,

a protocol constructor operable to accept data from said user interface and to construct an additional protocol in accordance therewith, said protocol being able to operate on digital data obtained via said interface,

a protocol selector for selecting any combination of said protocols to operate on said digital data and

an output for outputting results of applying said digital data to said selected protocols.

4. A digital simulation device comprising:

a plurality of predetermined protocols,

a user interface,

a protocol constructor operable to accept data from said user interface and to construct an additional protocol in accordance therewith, said protocol being able to operate on digital data obtained via said interface,

a protocol selector for selecting any combination of said protocols to operate on said digital data and

an output for outputting results of applying said digital data to said selected protocols.

5. A device according to claim 4, further comprising a script constructor for preparing scenarios for simulation.

6. A device according to either of claims 4 and 5, comprising further inputs to collect data from the simulation for testing.

7. A device according to any preceding claim, wherein said digital data is the output of all or part of a communications device.

8. A device according to any preceding claim, comprised within a computer.

9. A device according to any preceding claim, wherein said output comprises a configurable data filter.

10. A device according to any of claims 1 to 3 and 7 to 9, wherein said data interface comprises a selectable one of a plurality of specific interfaces.

11. A device according to either of claim 9 and claim 10, wherein said specific interfaces are designed for specific digital equipment and are operable to hide characteristics of the equipment from the device.

12. A device according to either of claims 10 and 11, further comprising an interface constructor operable through a user interface to construct a specific interface.

13. A device according to any preceding claim, operable with any digital data.

14. A device according to any preceding claim, wherein said protocols are operable to synthesize the operation of various electronic apparatus.

15. A device according to any preceding claim, wherein said protocols are templates for fitting said data into a data field structure of a communication protocol.

16. A device according to any preceding claim wherein said protocols are descendant objects of a protocol-independent parent object.

17. A device according to claim 16, wherein said protocol-independent parent object comprises a tree structure.

18. A device according to either of claim 16 and claim 17, wherein said protocol-independent parent object is operable to support processing of data according to any predefined protocol which is a descendant object thereof.

19. A device according to claim 16 or claim 18, wherein said protocol-independent parent object is operable to support any one of a group comprising smart filtering, triggering and simulation operations.

20. A device according to any one of claims 3 to 19, wherein said protocol constructor comprises a bank of predefined protocol substeps represented by visual items in the user interface, individually selectable by a user through said user interface to build a protocol.

21. A device according to either of claim 9 and claim 10, wherein said device interfaces are represented by visual items in the user interface, individually selectable by a user through said user interface.

22. A device according to any of claims 9, 10, and 21, comprising a plurality of additional device interfaces, allowing a plurality of devices to be tested simultaneously.

23. A device according to claim 22, wherein the number of additional device interfaces is limited only by constraints of available device memory.

24. A device according to any preceding claim, further having a storage unit for storing data for later analysis.

25. A device according to claim 22, wherein logical connections between different ones of said plurality of devices to be tested are viewable within said device.

26. A device according to claim 4, wherein a plurality of different simulations are operable to be run simultaneously and wherein said device is configurable to show logical connections between said simulations.

27. A device according to any preceding claims wherein a plurality of channels may have logical connections made therebetween and wherein a logical scenario is testable over said logical connections.

28. A device according to any preceding claims wherein a plurality of channels may have logical connections made therebetween and wherein a logical scenario is simulatable over said logical connections.

29. A device according to claim 22, wherein automatic association is made between a given device interface and one of said plurality of protocols.

30. A digital testing device substantially as hereinbefore described with reference to the accompanying drawings.

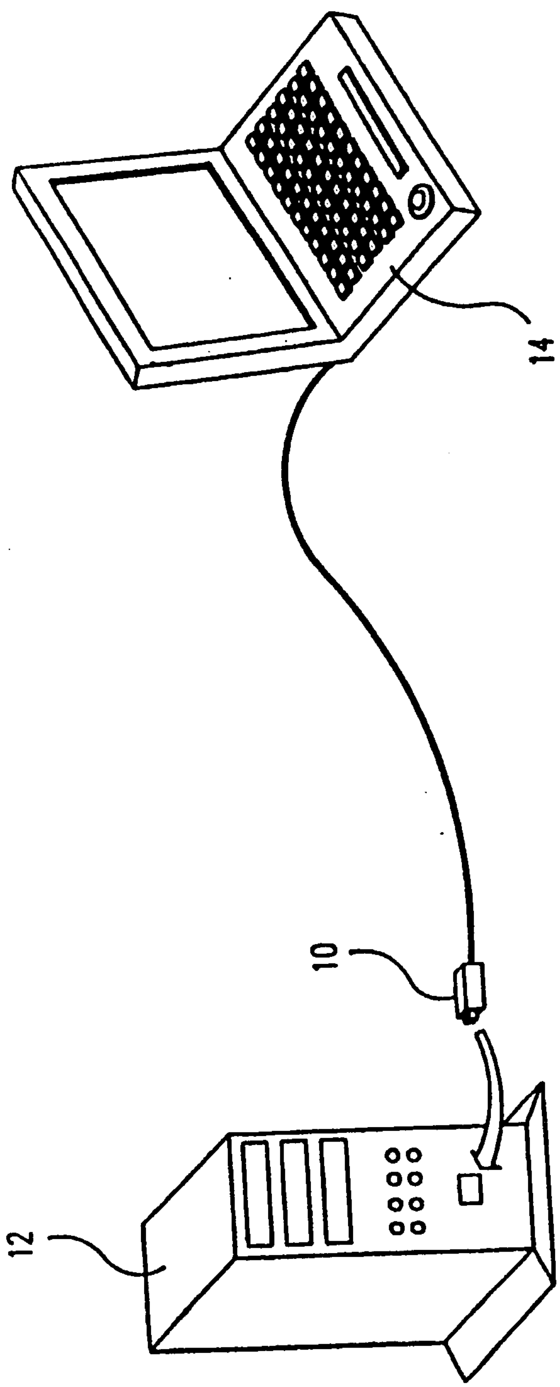


FIG. 1

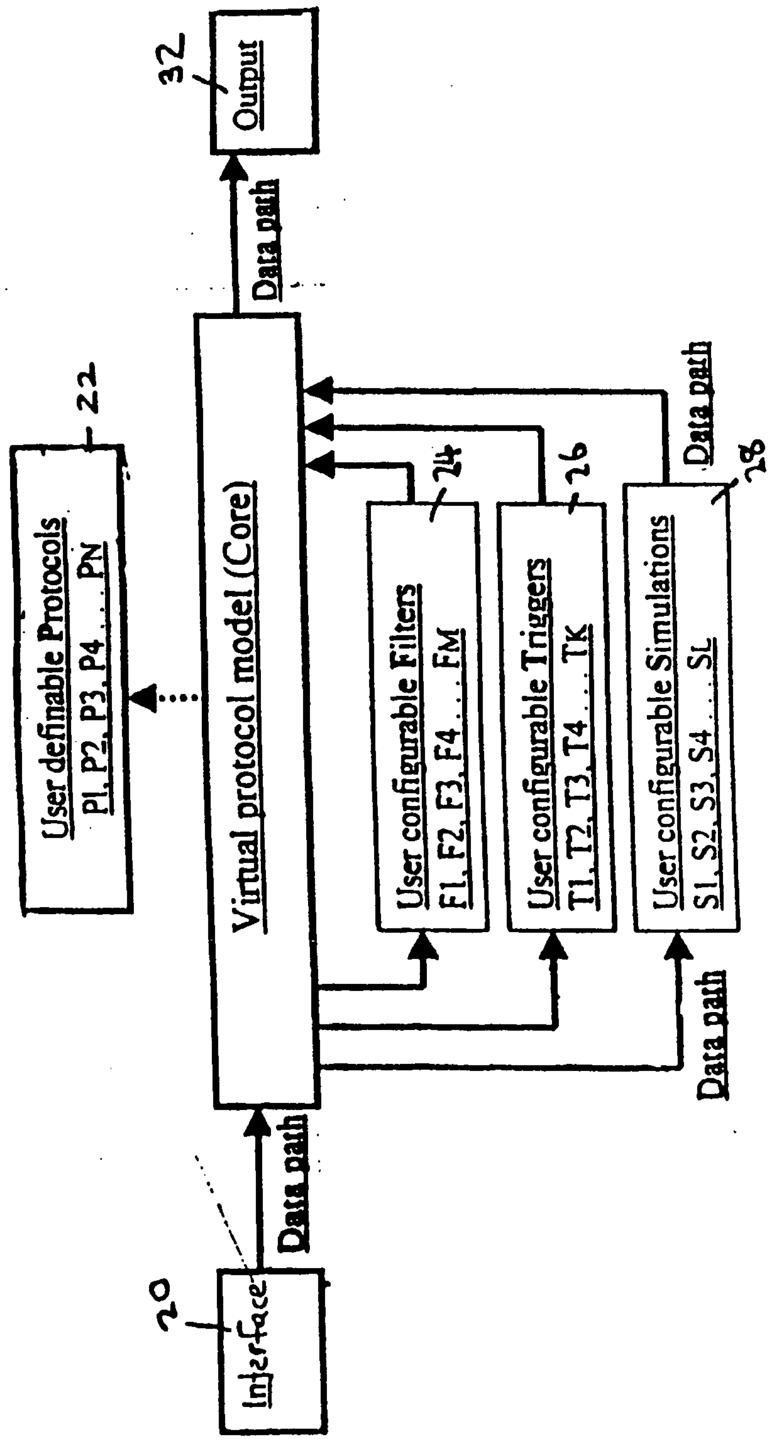

32
Output
Data path
User definable Protocols
P1, P2, P3, P4 . . . PN
22
Virtual protocol model (Core)
User configurable Filters
F1, F2, F3, F4 . . . FM
24
User configurable Triggers
T1, T2, T3, T4 . . . TK
26
User configurable Simulations
S1, S2, S3, S4 . . . SL
28
Data path
Data path
Data path
20
Interface

FIG. 2

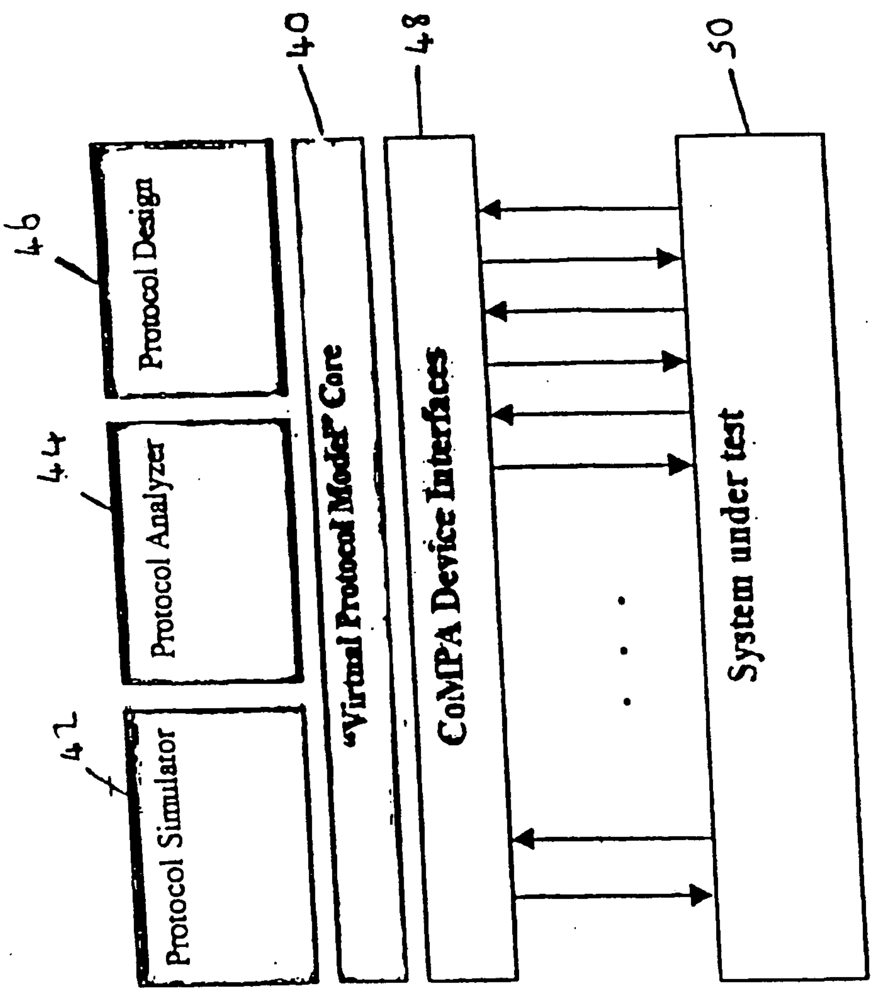

46
44
42
40
48
50
Protocol Design
Protocol Analyzer
Protocol Simulator
"Virtual Protocol Model" Core
CoMPA Device Interfaces
System under test

FIG. 3

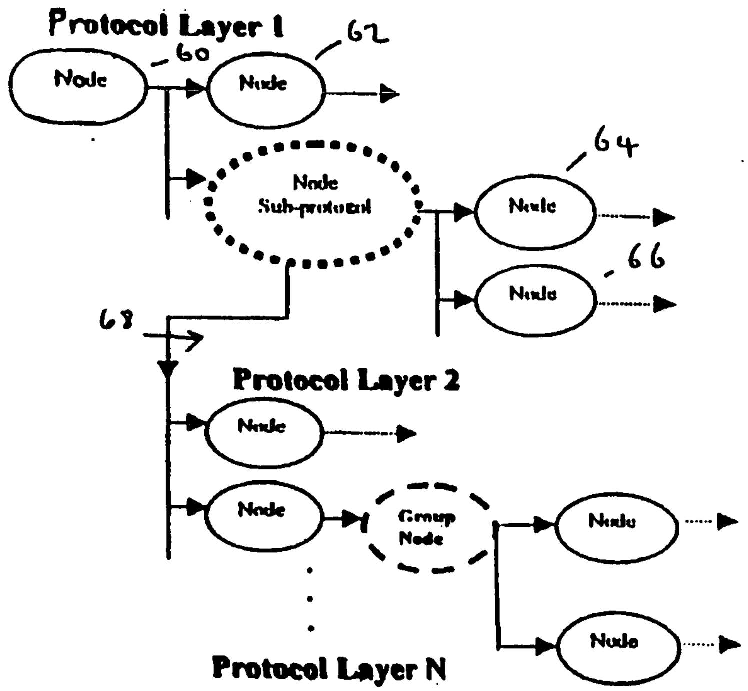
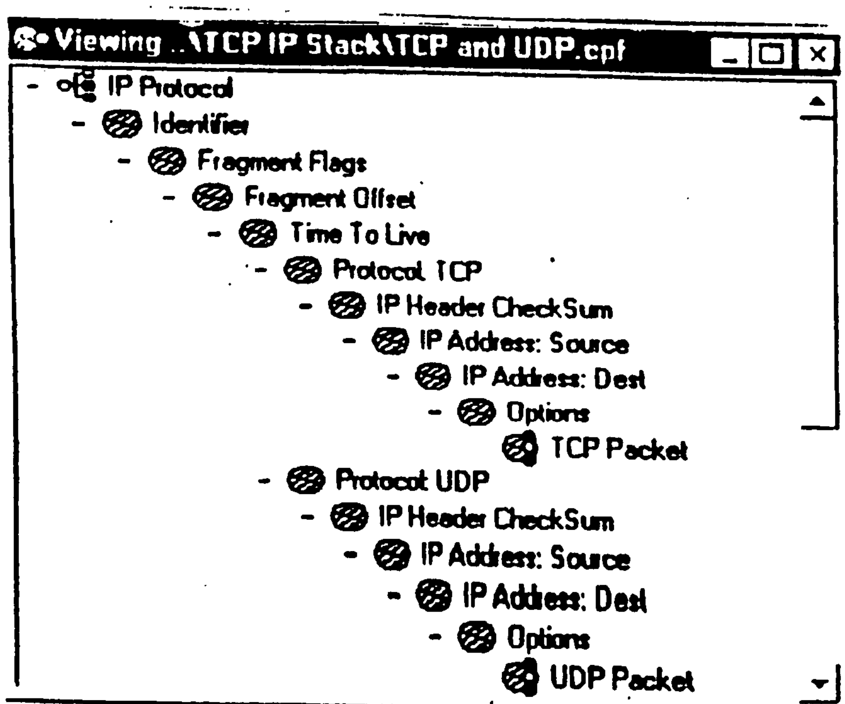


FIG. 4



Viewing a protocol in design utility

FIG. 5

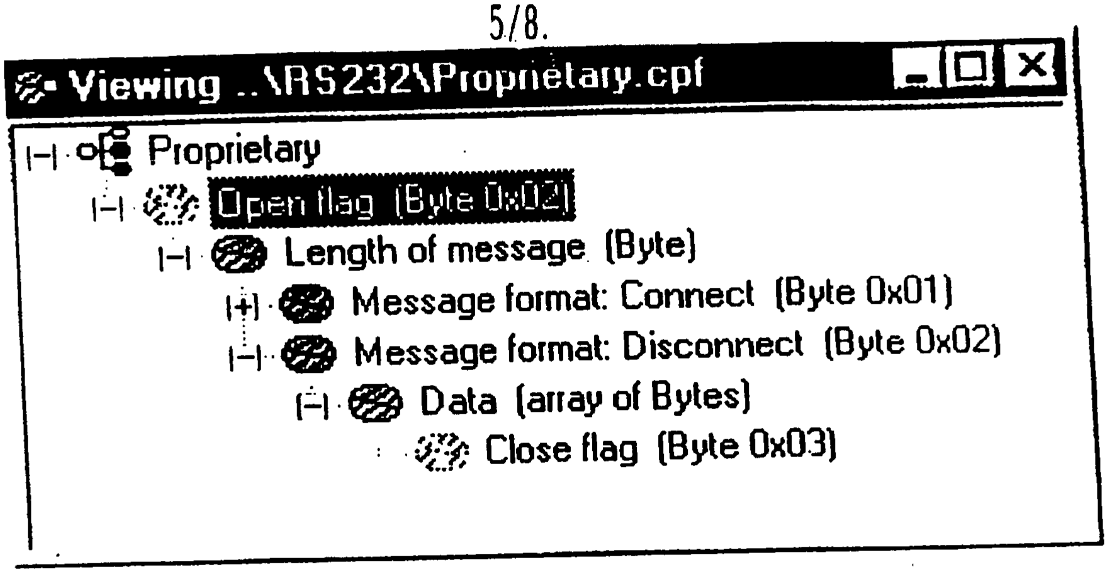


FIG 6

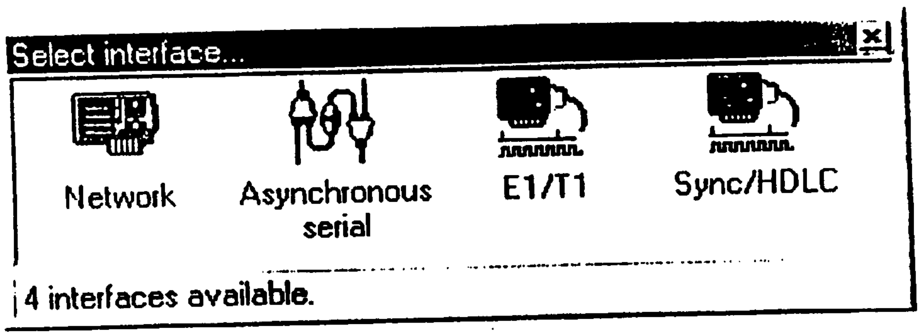


FIG 7

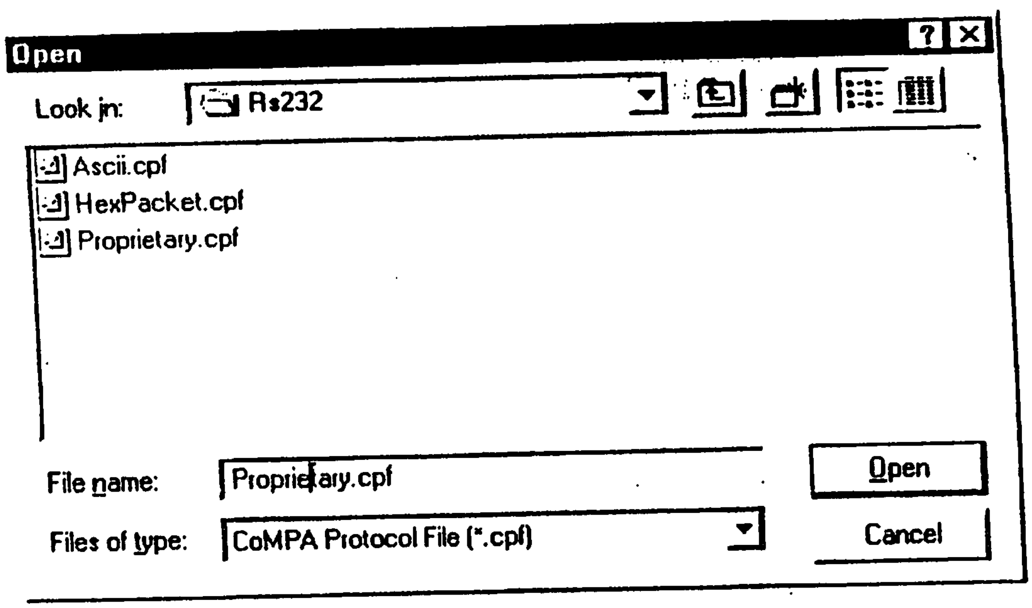


FIG 8

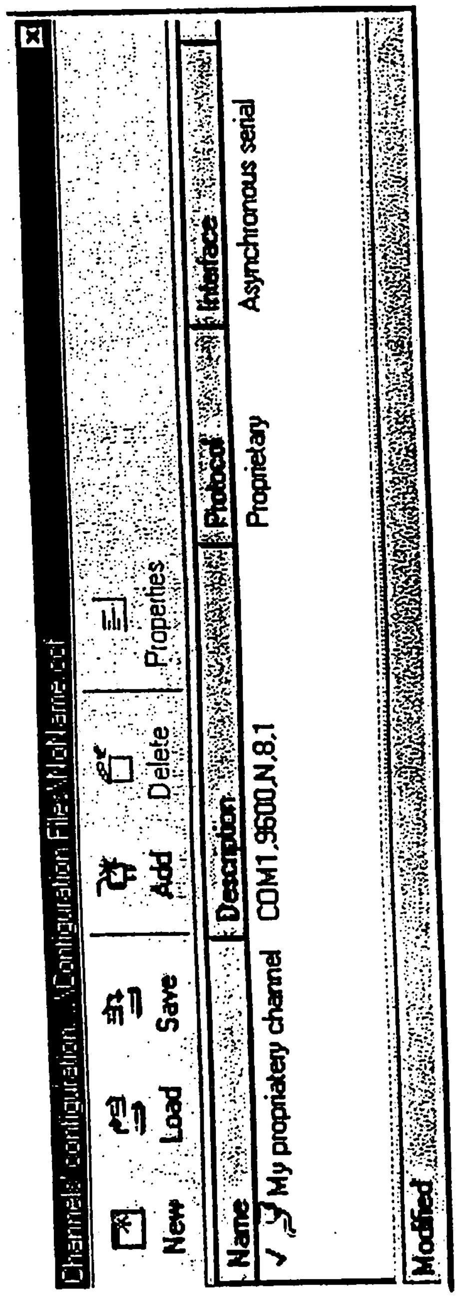

Channels' configuration ...\Configuration Files\NoName.ccf
New
Load
Save
Add
Delete
Properties
Name
Description
Protocol
Interface
My proprietary channel
COM1,9600,N,8,1
Proprietary
Asynchronous serial
Modified

FIG 9

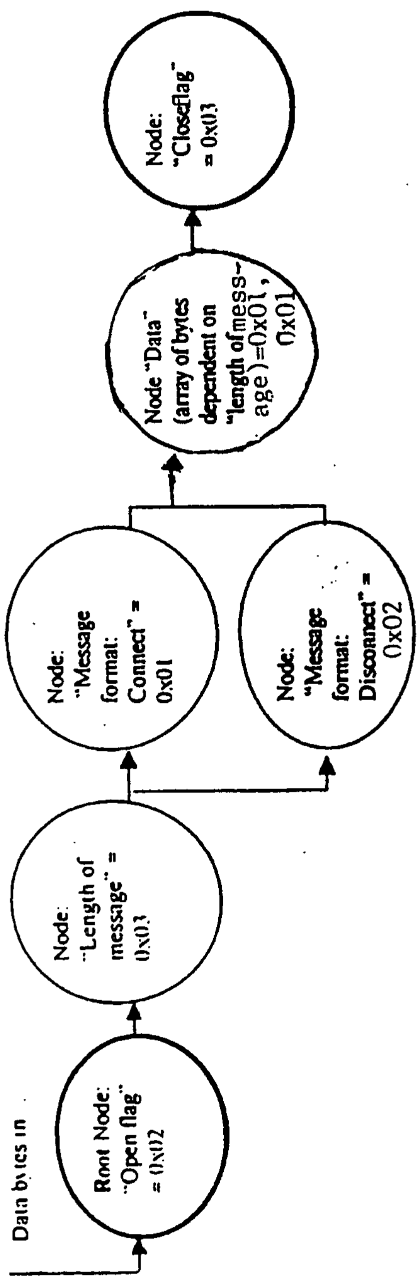

Data bytes in
Root Node: "Open flag" = 0x02
Node: "Length of message" = 0x03
Node: "Message format: Connect" = 0x01
Node: "Message format: Disconnect" = 0x02
Node "Data" (array of bytes dependent on "length of message)=0x01, 0x01
Node: "Closeflag" = 0x03

FIG 10

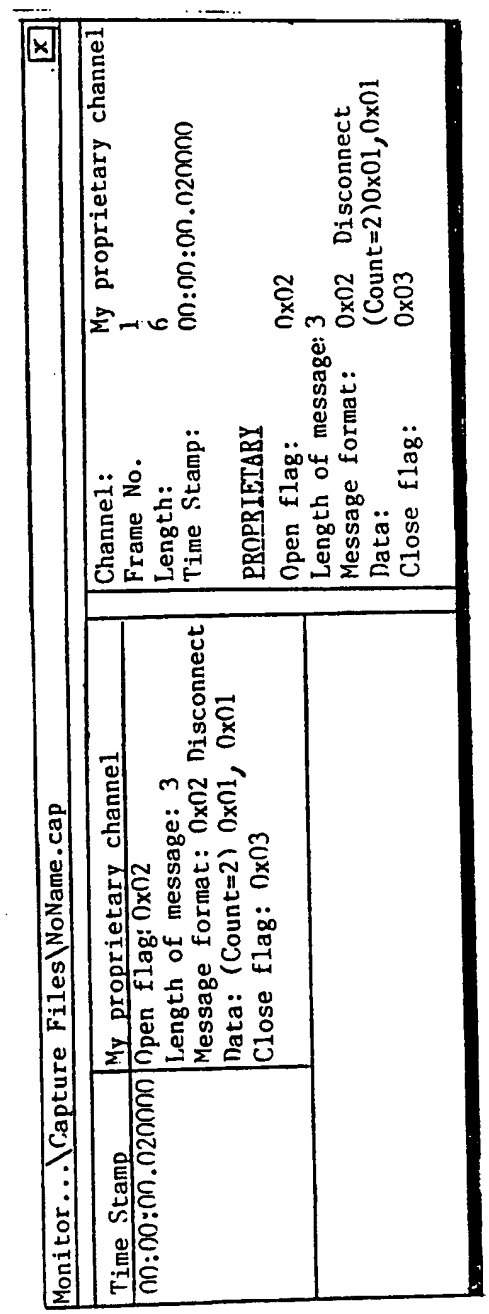

Monitor...\Capture Files\NoName.cap
Time Stamp
My proprietary channel
00:00:00.020000
Open flag: 0x02
Length of message: 3
Message format: 0x02 Disconnect
Data: (Count=2) 0x01, 0x01
Close flag: 0x03
Channel: My proprietary channel
Frame No. 1
Length: 6
Time Stamp: 00:00:00.020000
PROPRIETARY
Open flag: 0x02
Length of message: 3
Message format: 0x02 Disconnect
Data: (Count=2)0x01,0x01
Close flag: 0x03

FIG 11

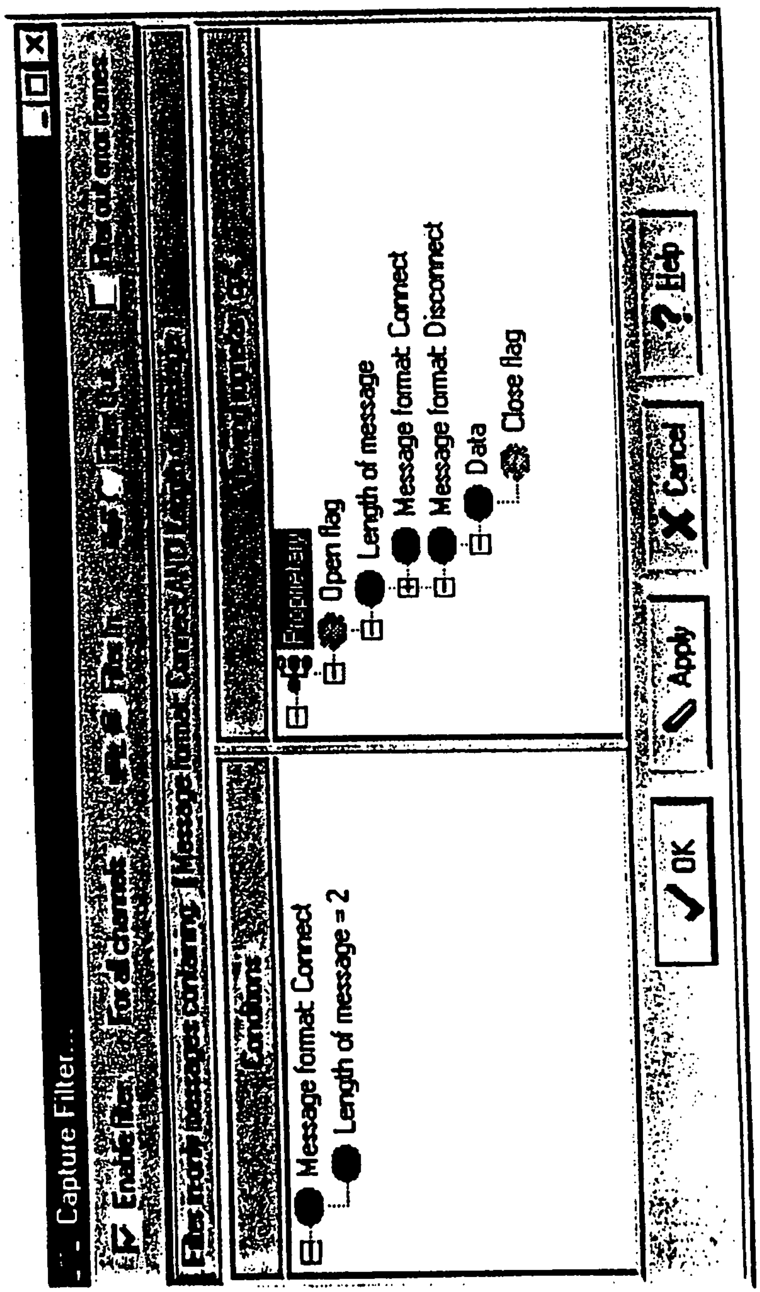

Capture Filter...
Conditions
Message format Connect
Length of message = 2
Parameters
Open flag
Length of message
Message format: Connect
Message format: Disconnect
Data
Close flag
OK
Apply
Cancel
Help

FIG 12

```
/*********************************************************
 * File name: protocol_analyze.h
 * Purpose:   CoMPA's virtual protocol anlysis core
 *
 * Author:    Comlog.
 * Version:   1.0 AUG,22,1999
 *********************************************************
 *      (C) Copyrights Comlog.LTD
 *********************************************************/

//-----------------------------------------------------------
#ifndef protocol_analyzeH
#define protocol_analyzeH

#include "protocol_defs.h"

class TChannelProtocolAnalyzer
{
  private:

    PT_PROTO_TREE  Layer1pProtocol;  // Lowest layer protocol tree
    UINT64,        TimeOut;          // Time out for frame resynchronization (in micro seconds)
    BYTE           *FrameBuf;        // captured data bytes buffer for frame.
    DWORD          MaxFrameSize;     // Maximum size of frame

    UINT64         StartFrameTime;   // Maintian start of frame timetag while receiveing frame
    DWORD          FrameSize;        // accumulated frame size.
    DWORD          FrameAnalyzedPos; // Total analyzed size from accumulated frame size
    DWORD          DecodeFuncIndex;  // Index in decoded data when using decoding functions.
    BYTE           DecodeFuncState;  // Last decoding function state.

    PT_PROTO_NODE  Layer1pNode;      // Last node or NULL for none (for layer 1 Protocol only).
    BOOLEAN        Layer1GroupRecurs; // Flag indicates whether inside a group recursive or not (for layer 1 Protocol only).
    DWORD          Layer1GroupMult;  // Group multiplies counter (for Lowest Protocol only).

    TOnSaveFrame   SaveFrameFunc;    // NULL or pointer to "save frame function"
    TOnDisplay     DisplayFunc;      // NULL or pointer to "Display node function"
    TPassFilterFunc PassFilterFunc;  // NULL or pointer to "Filter pass check function"

    BYTE           LogicalNumber;    // Logical Channel Number

    BYTE           NoProtocolBuffer[DEFAULT_BUF_SIZE]; // Default Buffer for "no protcol" situation

    void __fastcall StartNewFrame ( UINT64 TimeTag );
    void __fastcall AddFrame ( WORD ErrorTag, UINT64 TimeTag );

    void __fastcall AnalyzeLayer ( PT_PROTO_TREE pProtocol, BYTE *pData, DWORD DataSize, BYTE Layer );
    WORD __fastcall AnalyzeNode ( PT_PROTO_NODE pNode, BYTE *pData, DWORD *pDataIndex, DWORD DataSize,
BYTE CurrLayer );
    WORD __fastcall AnalyzeGroup ( PT_PROTO_NODE pNode, BYTE *pData, DWORD *pDataIndex, DWORD DataSize,
BYTE CurrLayer );
    PT_PROTO_NODE __fastcall MatchField ( PT_PROTO_NODE pNode, DWORD Multiples );
    bool __fastcall MatchBits ( PT_PROTO_FIELD pField, PT_BITS pBits, PT_SYNC_VALUE pSyncVal );
    WORD __fastcall DecodeField ( PT_PROTO_NODE pNode, BYTE *pData, DWORD *pDataIndex, DWORD DataSize,
DWORD *pMultiples );
    bool __fastcall MatchMultField ( PT_PROTO_FIELD pField, DWORD Multiples, PT_SYNC_VALUE pSyncVal );
    bool __fastcall MatchSingleField ( PT_PROTO_FIELD pField, PT_SYNC_VALUE pSyncVal, BYTE *pOutData );
    bool __fastcall DecodeAndConvertMultField ( PT_PROTO_FIELD pField, DWORD Multiples, BYTE *pInData );
    void __fastcall DecodeAndConvertSingleField ( PT_PROTO_FIELD pField, BYTE *pInData, BYTE *pOutData );
    void __fastcall CheckNodeConditions ( PT_PROTO_NODE pNode, DWORD Multiples );
    void __fastcall CheckProtocolConditions ( PT_PROTO_TREE pProtocol );
    DWORD __fastcall NodeValueAsDWORD ( PT_PROTO_NODE pNode );

  public:

    // constructor
    TChannelProtocolAnalyzer ( void );

    // Called upon initializing channel and binding it with a protocol
    void __fastcall Assign (  PT_PROTO_TREE   _Layer1pProtocol,
```

```
/*********************************************************
* File name: protocol_analyze.cpp
* Purpose:   CoMPA's virtual protocol anlysis core
*
* Author:    Comlog.
* Version:   1.0 AUG,22,1999
*********************************************************
*          (C) Copyrights Comlog.LTD
*********************************************************/

//-----------------------------------------------------------

#include "protocol_analyze.h"

extern DWORD __TotalFramesReceived;

/************************/
/***** P U B L I C *****/
/************************/

/**********************************************************************
*
* NAME:        TChannelProtocolAnalyzer::TChannelProtocolAnalyzer
*
* DESCRIPTION:  Constructor.
*
* RETURN VALUE:  None.
*
* NOTES:       None.
*
**********************************************************************/
TChannelProtocolAnalyzer::TChannelProtocolAnalyzer( void )
{
  Layer1pProtocol    = NULL;
  SaveFrameFunc      = NULL;
  DisplayFunc        = NULL;
  PassFilterFunc     = NULL;
  TimeOut            = 0; // No timeout !
  FrameBuf           = NoProtocolBuffer;
  MaxFrameSize       = sizeof(NoProtocolBuffer);
  StartFrameTime     = 0;
  FrameSize          = 0;
  FrameAnalyzedPos   = 0;
  DecodeFuncIndex    = 0;
  DecodeFuncState    = DEC_STATE_BEGIN;
  Layer1GroupRecurs  = FALSE;
  Layer1GroupMult    = 0;
  Layer1pNode        = NULL;
  LogicalNumber      = 0;
}

/**********************************************************************
*
* NAME:        TChannelProtocolAnalyzer::Assign
*
* DESCRIPTION:  Assign a protocol for using in this channel.
*              Also assign the interface' functions for filtering
*              and display.
*
* RETURN VALUE:  None.
*
* NOTES:       If DisplayFunc is NULL (No Display functionality at all)
*              If MatchFunc is NULL (No filters/triggers functionality
*              at all)
*
**********************************************************************/

void __fastcall TChannelProtocolAnalyzer::Assign ( PT_PROTO_TREE _Layer1pProtocol,
        TOnSaveFrame   _SaveFrameFunc,
```

```
      TOnDisplay     _DisplayFunc,
      TPassFilterFunc _PassFilterFunc,
      BYTE           _LogicalNumber )
{
 LayerIpProtocol   = _LayerIpProtocol;
 SaveFrameFunc     = _SaveFrameFunc;
 DisplayFunc       = _DisplayFunc;
 PassFilterFunc    = _PassFilterFunc;
 LogicalNumber     = _LogicalNumber;

 if ( LayerIpProtocol != NULL )
 {
   TimeOut       = LayerIpProtocol->TimeOut;
   FrameBuf      = LayerIpProtocol->FrameBuf;
   MaxFrameSize  = LayerIpProtocol->MaxFrameSize;
 }
 else
 {
   TimeOut       = 0;
   FrameBuf      = NoProtocolBuffer;
   MaxFrameSize  = sizeof(NoProtocolBuffer);
 }
}

/***********************************************************************
 *
 * NAME:        TChannelProtocolAnalyzer::Prepare
 *
 * DESCRIPTION:   Prepares channel for capture - reset all static parameters.
 *
 * RETURN VALUE:  None.
 *
 * NOTES:       None.
 *
 ***********************************************************************/
void __fastcall TChannelProtocolAnalyzer::Prepare ( UINT64 TimeTag )
{
  StartNewFrame ( TimeTag );
}

/***********************************************************************
 *
 * NAME:        TChannelProtocolAnalyzer::OnFinish
 *
 * DESCRIPTION:   Called upon capture finish. Saves remain frame's bytes.
 *
 * RETURN VALUE:  None.
 *
 * NOTES:       None.
 *
 ***********************************************************************/
void __fastcall TChannelProtocolAnalyzer::OnFinish ( void )
{
  if ( FrameSize > 0 )
  {
    // Just add remain bytes as garbage (FRAME_ERR_NOT_FINISHED):
    FrameAnalyzedPos = FrameSize;
    AddFrame( FRAME_ERR_NOT_FINISHED, StartFrameTime );
  }
}

/***********************************************************************
 *
 * NAME:        TChannelProtocolAnalyzer::OnPacket
 *
 * DESCRIPTION:   Process captured packet of bytes. This routine is the
 *              analysis entry point.
```

```
*
*        *Packet - Captured "chunk of bytes" (non synchronized).
*
*        PacketSize - Length of packet.
*
*        TimeTag - Interface/Driver Capture 64bit micro resolution
*             Time Tag.
*
* RETURN VALUE:  None.
*
* NOTES:      Called by Driver/Interface DLL high priority Thread.
*
*        Buffering Concept
*        ---------------
*
*        There are three buffering stages in data analysis:
*
*        1. Actual Bytes/Packet received - these bytes must be
*          handled as quicklly as possible and returned back to
*          driver for reuse.
*        2. Frame buffer - Accumulating data storage for completing
*          synchronized data frame.
*        3. Decoded Data - Decoded data bytes after process. These are
*          maintained per each node(field) in the protocol Tree.
*
**************************************************************************/

void __fastcall TChannelProtocolAnalyzer::OnPacket ( BYTE *Packet,
                         DWORD PacketSize,
                         WORD PacketErr,
                         UINT64 TimeTag )
{
  DWORD  PacketIndex = 0; // Position in packet.
  WORD   LastErr;    // Last error returned from analysis function
  bool   AnotherLoop = FALSE; // Set to TRUE if need extra anlysis loop

  /*************************************************************************/
  /* Check for timeout ( Check only if frame has alredy started ):      */
  /*------------------------------------------------------------     */
  /* If TimeOut >= 0: whole frame must be completed in an interval that is */
  /*           less then TimeOut !                  */
  /* If TimeOut = 0: whole frame must be received in single packet     */
  /*************************************************************************/

  if ( FrameSize )
  {
    // Notice: If TimeOut = 0 then even two successive packets with the
    // same time tag (Delta time = 0 ) are considered timeout.
    if ( TimeTag >= ( StartFrameTime + TimeOut ) )
    {
      AddFrame( FRAME_ERR_TIME_OUT, TimeTag ); // Error - Add all accumulated bytes (not including currently received)
with timeout tag
    }
    // else: Ok, valid state ( in the middle of a frame ).
  }
  else
    StartNewFrame ( TimeTag );

  // Keep going till whole packet' bytes proccessed:
  while ( ( PacketIndex < PacketSize ) || ( AnotherLoop ) )
  {
    AnotherLoop = FALSE;

    /*****************************************/
    /* Enough space left in frame buffer ?  */
    /*****************************************/

    if ( (MaxFrameSize - FrameSize) >= (PacketSize - PacketIndex) )
    {
      // Add remained packet bytes to frame:
```

```
      memcpy( &FrameBuf[FrameSize], &Packet[PacketIndex], (PacketSize - PacketIndex) );
      FrameSize += (PacketSize - PacketIndex);
      PacketIndex = PacketSize;
    }
    else
    {
      // Add maximum amount of bytes to fill frame' buffer:
      memcpy( &FrameBuf[FrameSize], &Packet[PacketIndex], (MaxFrameSize - FrameSize) );
      PacketIndex += (MaxFrameSize - FrameSize);
      FrameSize = MaxFrameSize;
    }

    if ( PacketErr ) // Packet Error ???
    {
      FrameAnalyzedPos = FrameSize; // Act as if analyzed...
      AddFrame( PacketErr, TimeTag ); // Add frame and prepare for next
    }
    else
    {
      if ( Layer1pProtocol != NULL )
        LastErr = AnalyzeNode ( Layer1pNode, FrameBuf/*&FrameBuf[FrameAnalyzedPos]*/, &FrameAnalyzedPos,
FrameSize, FIRST_LAYER );
      else
      {
        FrameAnalyzedPos = FrameSize; // Act as if analyzed...
        LastErr = 0; // No Error !
      }


      switch ( LastErr )
      {
        /*****************************************************************/
        /* Ok message complete and FrameAnalyzedPos = FrameSize        */
        /*****************************************************************/
        case 0 :   if ( TimeOut == 0 ) // No timeout
                     FrameAnalyzedPos = FrameSize;
                   AddFrame( 0, TimeTag ); // Add frame and prepare for next
                   break;

        /*****************************************************************/
        /* 1.Still more nodes to analyze - required more bytes.        */
        /* 2.Search pattern not found required more bytes.             */
        /* 3.In the middle of decoding function.                       */
        /*****************************************************************/
        case ANLZ_ERR_NODES_BUT_NOT_ENOUGH_DATA:
        case ANLZ_ERR_NOT_ENOUGH_DATA_TILL_END:
        case ANLZ_ERR_PATTERN_NOT_FOUND:
        case ANLZ_ERR_DFUNC_NOT_FINISHED:
                if ( FrameSize < MaxFrameSize ) // Fragmented message, wait for more data...
                {
                  if ( TimeOut == 0 ) // No timeout
                  {
                    FrameAnalyzedPos = FrameSize;
                    AddFrame( 0 , TimeTag ); // Add frame and prepare for next
                  }
                  break;
                }
                // Else, FrameSize = MaxFrameSize but still not enough data:
                AddFrame( FRAME_ERR_TOO_BIG, TimeTag ); // Add frame and prepare for next
                break;

        /*****************************************************************/
        /* Message complete. (yet still more bytes in frame buffer)    */
        /*****************************************************************/
        case ANLZ_ERR_DATA_BUT_NO_MORE_NODES:
                if ( TimeOut == 0 ) // No timeout so also add remain bytes
                  FrameAnalyzedPos = FrameSize;
                AddFrame( 0, TimeTag ); // Add frame and prepare for next
                if ( FrameAnalyzedPos < FrameSize ) // If fragmented message, wait for more data...
                  AnotherLoop = TRUE; // repeat another loop - still bytes in frame buffer
```

```
        /*********************************************************************/
        /* 1.Matching found could not be found !                */
        /* 2.Could not complete analysis since decoding buf was too small.*/
        /* 3.Decoding function returned error (out of synchronization).  */
        /*********************************************************************/
        case ANLZ_ERR_NO_MATCH_FOUND:
        case ANLZ_ERR_DECODE_BUF_TOO_SMALL:
        case ANLZ_ERR_DFUNC_OUT_OF_SYNC:
              if ( TimeOut == 0 ) // No timeout so also add remain bytes (Let re-anlysis detect errors)
              {
                FrameAnalyzedPos = FrameSize;
                AddFrame( 0, TimeTag ); // Add frame and prepare for next
              }
              else
                AddFrame( FRAME_ERR_OUT_OF_SYNC, TimeTag ); // Add frame and prepare for next
              if ( FrameAnalyzedPos < FrameSize ) // If fragmented message, wait for more data...
                AnotherLoop = TRUE; // repeat another loop - still bytes in frame buffer
              break;

        /*********************************************************************/
        /* Invalid Frame Check Sequece Returned from decoding function.  */
        /*********************************************************************/
        case ANLZ_ERR_DFUNC_INVALID_FCS:
              if ( TimeOut == 0 ) // No timeout so also add remain bytes
                FrameAnalyzedPos = FrameSize;
              AddFrame( FRAME_ERR_INVALID_FCS, TimeTag ); // Add frame and prepare for next
              if ( FrameAnalyzedPos < FrameSize ) // If fragmented message, wait for more data...
                AnotherLoop = TRUE; // repeat another loop - still bytes in frame buffer
              break;

        /* Unexpected !!! */
        default:   AddFrame( FRAME_ERR_UNEXPECTED, TimeTag ); // Add frame and prepare for next
      }
    }
  }
}

/**********************************************************************
 *
 * NAME:        TChannelProtocolAnalyzer::ReAnalyzeFrame
 *
 * DESCRIPTION:  Proccess an already captured frame for monitoring
 *               purpose.
 *
 *               Frame - Frame bytes buffer.
 *               FrameSize - Frame buffer size.
 *               _DisplayFunc - Call back for node/group display function.
 *
 * RETURN VALUE:  None.
 *
 * NOTES:        Called by monitor for analyzing non-error frames.
 *
 **********************************************************************/

void __fastcall TChannelProtocolAnalyzer::ReAnalyzeFrame ( BYTE *pFrame,
                                   DWORD FrameSize,
                                   TOnDisplay _DisplayFunc,
                                   TPassFilterFunc _PassFilterFunc )
{
  StartNewFrame ( 0 ); // Reset all channel's static variabels

  DisplayFunc     = _DisplayFunc;
  PassFilterFunc  = _PassFilterFunc;

  if ( Layer1pProtocol != NULL )

    // Returned with error ?
```

6

```
  if( AnalyzeNode ( Layer1pNode, pFrame , &FrameAnalyzedPos, FrameSize, FIRST_LAYER ) ==
ANLZ_ERR_DATA_BUT_NO_MORE_NODES )
   {
    if( DisplayFunc != NULL )
      DisplayFunc ( 1, NULL, &pFrame[FrameAnalyzedPos], FrameSize - FrameAnalyzedPos, 0,
ANLZ_ERR_DATA_BUT_NO_MORE_NODES );
   }
}

/************************************/
/***** P R I V A T E      *****/
/************************************/

/*********************************************************************
 *
 * NAME:        TChannelProtocolAnalyzer::StartNewFrame
 *
 * DESCRIPTION:   Reset all required parameters for starting anlysis of
 *          new frame ( applied on layer1 protocol only).
 *
 * RETURN VALUE:  None.
 *
 * NOTES:       None.
 *
 *********************************************************************/
void __fastcall TChannelProtocolAnalyzer::StartNewFrame ( UINT64 TimeTag )
{
  StartFrameTime     = TimeTag;      // Set time ( Starting new frame )
  FrameSize         = 0;          // No bytes yet
  FrameAnalyzedPos   = 0;          // No bytes yet
  DecodeFuncIndex    = 0;          // Nothing decoded yet
  DecodeFuncState    = DEC_STATE_BEGIN; // Initial state
  Layer1GroupRecurs  = FALSE;        // Not inside a group recursive
  Layer1GroupMult    = 0;          // No multiples of group

  /*NumOfFrameNodes    = 0;
  for ( i = 0; i < MAX_CONDITIONS_TYPES_PER_NODE; i++ )
    NumOfConditions[i] = 0;*/

  if( Layer1pProtocol != NULL )
    Layer1pNode       = Layer1pProtocol->pRootNode;
}
/*********************************************************************
 *
 * NAME:      AddFrame
 *
 * DESCRIPTION:  Add frame with/without error tag.
 *
 *          ErrorTag - Frame Error Value or 0 for no error.
 *          TimeTag - New Time tag for next frame.
 *
 * RETURN VALUE:  None.
 *
 * NOTES:      None.
 *
 *********************************************************************/
void __fastcall TChannelProtocolAnalyzer::AddFrame ( WORD ErrorTag, UINT64 TimeTag )
{
  DWORD Tmp;

  if( ErrorTag )
  {
    if( ( ErrorTag == FRAME_ERR_TOO_BIG ) ||
      ( ErrorTag == FRAME_ERR_TIME_OUT ) ||
      ( ErrorTag == FRAME_ERR_UNEXPECTED ) )
    {
      FrameAnalyzedPos = FrameSize; // Act as if bytes where analyzed !
    }
```

```
}
_TotalFramesReceived++; // Increase external receive counter (for monitoring purpose)

if ( SaveFrameFunc != NULL ) // Add to memory/file...
{
  if ( FrameAnalyzedPos == 0 )
    DEBUG_ERR ( "Before SaveFrameFunc with Zero size frame" );
  if ( PassFilterFunc != NULL ) // filter exists...
  {
    // Check if pass filter...
    if ( PassFilterFunc ( ErrorTag ) )
      SaveFrameFunc ( LogicalNumber, FrameBuf, FrameAnalyzedPos, ErrorTag, StartFrameTime );
  }
  else
    SaveFrameFunc ( LogicalNumber, FrameBuf, FrameAnalyzedPos, ErrorTag, StartFrameTime );
}

// Check for bytes in frame buffer (which are not part of message):
if ( FrameAnalyzedPos < FrameSize )
{
  Tmp = FrameSize - FrameAnalyzedPos;
  // memmove is neccessary - since may copy overlapping areas.
  // add remain bytes too new frame...
  memmove ( &FrameBuf[0], &FrameBuf[FrameAnalyzedPos], FrameSize - FrameAnalyzedPos );
  StartNewFrame ( TimeTag );
  FrameSize = Tmp;
}
else
  StartNewFrame ( TimeTag );
}

/**************************************************************************
 *
 * NAME:        AnalyzeLayer
 *
 * DESCRIPTION:   Analysis of higher protocol layer (higher layer then the
 *                previous)
 *
 *                pProtocol - new protocol layer,
 *                pData -    Decoded data passed from lower protocol layer
 *                DataSize - Decoded data size passed from lower protocol layer.
 *                Layer -    Previous protocol layer Number.
 *
 * RETURN VALUE:  None.
 *
 * NOTES:        This routine may be called recursivly from AnalyzeNode.
 *               It must not be used on first layer !
 *
 *               Notice that previous (lower) layer is totally independednt with
 *               this layer, hence this function returns nothing.
 *
 **************************************************************************/
void __fastcall TChannelProtocolAnalyzer::AnalyzeLayer ( PT_PROTO_TREE pProtocol, BYTE *pData, DWORD DataSize,
BYTE Layer )
{
  DWORD DataIndex = 0; // Starting new layer - so point to first posistion

  Layer++; // Increase layer number.

  if ( pProtocol->pRootNode == NULL )
    return;

  // Check filter/trigger conditions...
  if ( pProtocol->pFirstHookedCondition != NULL )
      CheckProtocolConditions ( pProtocol );

  /******************************************/
  /* Recursive analysis in protocol tree... */
```

```
/****************************************/
if ( AnalyzeNode ( pProtocol->pRootNode, pData, &DataIndex, DataSize, Layer ) ==
ANLZ_ERR_DATA_BUT_NO_MORE_NODES )
  {
   if ( DisplayFunc != NULL )
      DisplayFunc ( Layer, NULL, &pData[DataIndex], DataSize - DataIndex, 0,
ANLZ_ERR_DATA_BUT_NO_MORE_NODES );
  }

  /*****************************************************************/
  /*                                                              */
  /* This layer is complete:                                      */
  /* ---------------------                                        */
  /* Make sure all clear for analysis of more nodes on previous layer: */
  /* Reset decoding function state if left in the middle.          */
  /*****************************************************************/
  DecodeFuncState = DEC_STATE_BEGIN;
  DecodeFuncIndex = 0;
 }


/*******************************************************************
 *
 * NAME:        TChannelProtocolAnalyzer::AnalyzeNode
 *
 * DESCRIPTION:   Analysis and match for a specific node.
 *
 *          pNode -    pointer of next node to be analyzed.
 *          pData -    Received data' bytes buffer.
 *          pDataIndex - In: current index in Received data' bytes buffer.
 *                    Out: after analysis index in Received data' bytes buffer.
 *          DataSize -  Received data' bytes buffer size.
 *          CurrLayer - Current Protocol layer number 1..MAX_NUM_OF_LAYERS
 *
 * RETURN VALUE:  0 or analysis' error number.
 *
 * NOTES:        This routine is called recursivly.
 *
 *******************************************************************/

WORD __fastcall TChannelProtocolAnalyzer::AnalyzeNode ( PT_PROTO_NODE pNode, BYTE *pData, DWORD *pDataIndex,
DWORD DataSize, BYTE CurrLayer )
{
  WORD LastErr;         // Error Result from DecodeField function
  DWORD Multiples;      // Multiples of field/group
  DWORD PrevDataIndex;  // Previous data index (before analysis)

  if ( ( CurrLayer == FIRST_LAYER ) && ( TimeOut != 0 ) )
    Layer1pNode = pNode; // Maintain Node pointer (requires if data is fragmented)

  /**********************************/
  /* Check: Is called from a leaf ? */
  /**********************************/
  if ( pNode == NULL ) // If called from a leaf:
  {
    if ( *pDataIndex < DataSize ) // Still bytes to analyze:
    {
      return ( ANLZ_ERR_DATA_BUT_NO_MORE_NODES );
    }
    // Else: ( *pDataIndex == DataSize )
    return ( 0 ); // Ok, no more nodes to analyze.

  }

  /******************************************/
  /* Check: Is this node a Field ?          */
  /******************************************/
  if ( pNode->pField != NULL )
  {
```

```
PT_PROTO_FIELD pField;

pField = pNode->pField;

/*****************************************/
/* Decoding data and checking for error: */
/*****************************************/
PrevDataIndex = *pDataIndex;

LastErr = DecodeField ( pNode, pData, pDataIndex, DataSize, &Multiples );
if ( LastErr )
{
   // If data is not fragmented allow situation where still node to analyze but
   // no more data:
   if ( ( ( LastErr == ANLZ_ERR_NODES_BUT_NOT_ENOUGH_DATA ) &&
        ( *pDataIndex == DataSize ) ) &&
      ( ( CurrLayer != FIRST_LAYER ) || ( TimeOut == 0 ) ) )
     return ( 0 ); // Ok, pherhaps not all nodes anlayzed, yet, no more data !

   // Call Display with error...
   if ( DisplayFunc != NULL )
     DisplayFunc ( CurrLayer, pNode, &pData[PrevDataIndex], DataSize - PrevDataIndex, 0, LastErr );

   return ( LastErr );
 }

 /*******************************************/
 /* Try to match one of the sibling nodes  */
 /*******************************************/
 pNode = MatchField ( pNode, Multiples );

 if ( pNode == NULL ) // If no match:
 {
   if ( ( CurrLayer == FIRST_LAYER ) && ( TimeOut != 0 ) ) // Maintain Node pointer (requires if data is fragmented)
     Layer1pNode = NULL;

   // Call Display with error - no match !
   if ( DisplayFunc != NULL )
     DisplayFunc ( CurrLayer, pNode, &pData[PrevDataIndex], DataSize - PrevDataIndex, 0,
ANLZ_ERR_NO_MATCH_FOUND );

   return ( ANLZ_ERR_NO_MATCH_FOUND );
 }

 // Ok match: check trigger/filter...
 if ( pNode->pFirstHookedCondition != NULL )
   CheckNodeConditions ( pNode, Multiples );

 // Ok, Call Display...
 if ( DisplayFunc != NULL )
   DisplayFunc ( CurrLayer, pNode, &pData[PrevDataIndex], *pDataIndex - PrevDataIndex, Multiples, 0 );

 /*******************************************/
 /* Process higher protocol layer if such: */
 /*******************************************/
 if ( ( pNode->pHigherProto != NULL ) && ( Multiples > 0 ) )
   AnalyzeLayer ( pNode->pHigherProto , pField->DecodedBuf, Multiples*(pField->Type), CurrLayer );

 /*******************************************/
 /* Recursive call to child node (if such): */
 /*******************************************/
 if ( ( pNode->pChild == NULL ) && ( pNode->pOwnerGroupNode != NULL ) ) // Last was a leaf of a group:
 {
   // Check if first protocol layer and currently inside a recursive initiated from the group:
   if ( ( CurrLayer == FIRST_LAYER ) && ( TimeOut != 0 ) )
   {
     Layer1GroupMult++;

     if ( Layer1GroupRecurs ) // Called recursivly from a group:
     {
```

```
            LayerIpNode = pNode->pOwnerGroupNode; // Maintain Node pointer (requires if data is fragmented)
            return ( 0 ); // Ok.
         }
         else // Not called recursively, hence need to continue as if next node is the owner group:
            return ( AnalyzeGroup ( pNode->pOwnerGroupNode, pData, pDataIndex, DataSize, CurrLayer ) );
      }

      // Else:
      return ( 0 ); // Ok, Not the first layer (so defenetlly called recursively from a group)
   }

   // Else, Child exist (or no child but not part of group):
   return ( AnalyzeNode ( pNode->pChild, pData, pDataIndex, DataSize, CurrLayer ) );
}
/******************************************/
/* Check: Is this node a Group ?       */
/******************************************/
else
   if ( pNode->pGroup != NULL )
      return ( AnalyzeGroup ( pNode, pData, pDataIndex, DataSize, CurrLayer ) );

// Else ( for debug check):
DEBUG_ERR ( " Neither Field Nor Group ! ");
return ( ANLZ_ERR_UNEXPECTED );
}

/********************************************************************
 *
 * NAME:        TChannelProtocolAnalyzer::AnalyzeGroup
 *
 * DESCRIPTION:  Analysis of a group on first layer only.
 *
 *          pNode -   pointer of next node to be analyzed.
 *          pData -   Received data' bytes buffer.
 *          pDataIndex - In: current index in Received data' bytes buffer.
 *                  Out: after analysis index in Received data' bytes buffer.
 *          DataSize -  Received data' bytes buffer size.
 *          CurrLayer - Current Protocol layer number 1..MAX_NUM_OF_LAYERS
 *
 * RETURN VALUE:  0 or analysis' error number.
 *
 * NOTES:      This may be called recursivly ( but not if CurrLayer = 1 )
 *          This function should be invoked only if node is a group !
 *
 ********************************************************************/

WORD __fastcall TChannelProtocolAnalyzer::AnalyzeGroup ( PT_PROTO_NODE pNode, BYTE *pData, DWORD *pDataIndex,
DWORD DataSize, BYTE CurrLayer )
{
   PT_PROTO_GROUP pGroup;
   WORD LastErr = 0;
   DWORD Multiples;    // Multiples of group

   if ( ( CurrLayer == FIRST_LAYER ) && ( TimeOut != 0 ) ) // Need to maintain some variables (solve receive fragmentation)
   {
      LayerIpNode = pNode;    // Mark last node for frame fragmentation
      if ( pNode->pOwnerGroupNode != NULL ) // (Debug checks)
      {
         DEBUG_ERR ( "Group Inside Group is not allowed in first layer when timeout is not 0 !");
         return (ANLZ_ERR_UNEXPECTED);
      }

      pGroup = pNode->pGroup;

      /*********************************/
      /* Determine multiples of Group : */
      /*********************************/

      switch( pGroup->Mult )
      {
```

```
case MF_NO_MULT:   Multiples = 1;
          break;
case MF_SPECIFIED: Multiples = pGroup->MultParameter;.
          break;
case MF_NODE:      Multiples = NodeValueAsDWORD ( pGroup->pMultNode ) + pGroup->MultParameter;
          break;
case MF_TILL_END:  DEBUG_ERR ( "Repeat till end in first layer and timeout is not 0 - not supported !");
          return ( ANLZ_ERR_UNEXPECTED );
default :       DEBUG_ERR ( "Unknown Group' Mult !");
          return ( ANLZ_ERR_UNEXPECTED );
}

/*****************************************************************/
/* Notice: In first protocol layer reception is fragmented, hecne */
/* this function may return without finishing group analysis.    */
/*****************************************************************/

if ( Layer1GroupMult == 0 )
{
  // Match by default (since entered group):
  // Call trigger/filter...
  if ( pNode->pFirstHookedCondition != NULL )
    CheckNodeConditions ( pNode, Multiples );

}

Layer1GroupRecurs = TRUE;  // Steping into a group recursive

/******************************************************/
/* keep Analysis while 1. Need more multiplies of group. */
/*             2. No error.                 */
/*             3. Still more data to analyze.  */
/******************************************************/
while ( ( Layer1GroupMult < Multiples ) &&
    ( LastErr == 0 ) &&
    ( *pDataIndex < DataSize ) )
      LastErr = AnalyzeNode ( pGroup->pRootNode, pData, pDataIndex, DataSize, CurrLayer );

Layer1GroupRecurs = FALSE;  // Out of a layer1 group recursive

if ( Layer1GroupMult == Multiples )
{
  Layer1GroupMult = 0;        // reset (since finished group multiples)

  // Go to child...
  return ( AnalyzeNode ( pNode->pChild, pData, pDataIndex, DataSize, CurrLayer ) );
}

return ( LastErr );
}
else // Not the first layer or 0 timeout:
{
  int i = 0;

  pGroup = pNode->pGroup;

  // Match by default (since entered group):
  // Call trigger/filter...
  if ( pNode->pFirstHookedCondition != NULL )
    CheckNodeConditions ( pNode, Multiples );

  /********************************/
  /* Determine multiples of Group : */
  /********************************/

  switch( pGroup->Mult )
  {
    case MF_NO_MULT:   Multiples = 1;
             break;
    case MF_SPECIFIED: Multiples = pGroup->MultParameter;
```

```
            break;
    case MF_NODE:      Multiples = NodeValueAsDWORD ( pGroup->pMultNode ) + pGroup->MultParameter;
            break;

    case MF_TILL_END:

      i = DataSize - ( *pDataIndex ) + pGroup->MultParameter/*Nagative value*/;

      while ( i > 0 ) // Loop till reach end - delta
      {
        LastErr = AnalyzeNode ( pGroup->pRootNode, pData, pDataIndex, DataSize, CurrLayer );
        if ( LastErr )
        {
          if ( LastErr != ANLZ_ERR_DATA_BUT_NO_MORE_NODES ) // This is not an error - since need to repeat
group
            return ( LastErr );
        }
        i = DataSize - ( *pDataIndex) + pGroup->MultParameter/*Nagative value*/;
      }

      if (-i < 0 ) // Too many bytes analyzed !
      {
        // ####### Call Display with error - no more data but still left multiples ! put as hex...
        return ( ANLZ_ERR_NODES_BUT_NOT_ENOUGH_DATA );
      }

      // Go to child...
      return ( AnalyzeNode ( pNode->pChild, pData, pDataIndex, DataSize, CurrLayer ) );

    default :        DEBUG_ERR ( "Unknown Group' Mult !");
               return ( ANLZ_ERR_UNEXPECTED );
  }

  /***********************************************/
  /* Handles: MF_NO_MULT, MF_SPECIFIED, MF_NODE : */
  /***********************************************/

  while ( ( i < (int)Multiples ) &&
       ( *pDataIndex < DataSize ) )
  {
    LastErr = AnalyzeNode ( pGroup->pRootNode, pData, pDataIndex, DataSize, CurrLayer );
    if ( LastErr )
      return ( LastErr );
    else
      i++;
  }

  if ( i < (int)Multiples ) // Not enough bytes !
  {
    // ####### Call Display with error - no more data but still left multiples ! put as hex...
    return ( ANLZ_ERR_NODES_BUT_NOT_ENOUGH_DATA );
  }

  // Go to child...
  return ( AnalyzeNode ( pNode->pChild, pData, pDataIndex, DataSize, CurrLayer ) );
  }
}


/************************************************************************
 *
 * NAME:       TChannelProtocolAnalyzer::MatchField
 *
 * DESCRIPTION:   Trys to match field's decoded data in one of the sibling
 *           nodes (including this node).
 *
 *           pNode -    pointer to first node (in sibling list).
 *           Multiples - Multiples of decoded field.
 *
```

```
/*****************************************************/
/* For each multiples type:                        */
/* ----------------------                          */
/* 1.Check if enough data to complete field if not */
/*   return error.                                 */
/* 2.Decode/convert recived data.                  */
/* 3.increament DataIndex to next field.           */
/*****************************************************/

case MF_NO_MULT:  if ( pNode->pBits != NULL ) // Bits partitioning ?
          {
            /**************************************/
            /* Check for bits partition ( allowed */
            /* only if MF_NO_MULT )               */
            /* Perfrom decoding only if:          */
            /* First node in bits partitioning    */
            /**************************************/

            *pMultiples = 1;

            if ( pNode->pBits->pFirst == NULL ) // Is first ?
            {
              if ( (DWORD)DataSize - DataIndex >= (DWORD)Type ) // enough bytes ?
              {
                DecodeAndConvertSingleField ( pField, &pData[DataIndex], pField->DecodedBuf );
                (*pDataIndex) += Type;
              }
              else
                return ( ANLZ_ERR_NODES_BUT_NOT_ENOUGH_DATA );
            }
            // Ok ! decode ok, or not the first bits partition.
          }
          else // No bits partitioning:
          {
            if ( (DWORD)(DataSize - DataIndex) >= (DWORD)Type )
            {
              if ( ( pNode->pBits == NULL ) ||
                 ( ( pNode->pBits != NULL ) && ( pNode->pBits->pFirst == NULL ) ) )
              {
                DecodeAndConvertSingleField ( pField, &pData[DataIndex], pField->DecodedBuf );
                (*pDataIndex) += Type;
                *pMultiples = 1;
              }
            }
            else
              return ( ANLZ_ERR_NODES_BUT_NOT_ENOUGH_DATA );
          }

          break; // Ok

case MF_SPECIFIED: *pMultiples = pField->MultParameter;
          if ( DataSize - DataIndex >= ((*pMultiples)*Type) )
          {
            if ( !DecodeAndConvertMultField ( pField, *pMultiples, &pData[DataIndex] ) )
            {
              return ( ANLZ_ERR_DECODE_BUF_TOO_SMALL );
            }
            (*pDataIndex) += ((*pMultiples)*Type);
          }
          else
            return ( ANLZ_ERR_NODES_BUT_NOT_ENOUGH_DATA );
          break;

case MF_NODE:    *pMultiples = NodeValueAsDWORD ( pField->pMultNode ) + pField->MultParameter;
          if ( DataSize - DataIndex >= ((*pMultiples)*Type) )
          {
            if ( !DecodeAndConvertMultField ( pField, *pMultiples, &pData[DataIndex] ) )
            {
              // DEBUG_ERR ( " DecodeField: MF_NODE - but decode buf too small ! ");
              return ( ANLZ_ERR_DECODE_BUF_TOO_SMALL );
```

```
            }
            (*pDataIndex) += (pField->Type) * (*pMultiples);
        }
        else
            return ( ANLZ_ERR_NODES_BUT_NOT_ENOUGH_DATA );
        break;

/**********************************************************/
/* The following types applied only for TF_BYTE Type only */
/**********************************************************/

case MF_TILL_END:  if ( (int)(DataSize - DataIndex) < -(pField->MultParameter/*Nagative value*/) ) // Not enough bytes ! ( MultParameter is -delta from end )
            {
              *pMultiples = 0;
              return ( ANLZ_ERR_NOT_ENOUGH_DATA_TILL_END );
            }

            *pMultiples = DataSize - ( DataIndex ) + pField->MultParameter/*Nagative value*/;
            if ( !DecodeAndConvertMultField ( pField, *pMultiples, &pData[DataIndex] ) )
            {
              return ( ANLZ_ERR_DECODE_BUF_TOO_SMALL );
            }

            (*pDataIndex) = DataSize + pField->MultParameter;
            break;

case MF_FIND:   /**********************************************************/
            /* pDataIndex will be updated only if pattern was found.  */
            /* For fragmented data, search will be made again and again*/
            /* till pattern found or error (timeout or frame to big)  */
            /**********************************************************/

            *pMultiples = 0;
            while ( DataIndex < DataSize )
            {
              DecodeAndConvertSingleField ( pField, &pData[DataIndex], &pField->DecodedBuf[*pMultiples] );
              DataIndex++;
              if ( pField->DecodedBuf[*pMultiples] == (BYTE)pField->MultParameter )
              {
                (*pMultiples)++;
                *pDataIndex = DataIndex;
                return ( 0 ); // Ok, decoding success.
              }
              (*pMultiples)++;
            }

            return ( ANLZ_ERR_PATTERN_NOT_FOUND );

case MF_DECODE:   /**************************************************************************/
            /* External decoding function involved: Decode and increament pDataIndex.  */
            /* DecodeFuncState is static - if last time decoding function hasn't been  */
            /* finished, this variable maintain the last state.                       */
            /* Notice: single channel cannot use two decoding function processing     */
            /* in parallel !                                                          */
            /**************************************************************************/

            // Debug check:
            if ( pField->pDecode->pDecodeFunc == NULL )
            {
              DEBUG_ERR("No decoding function !");
              return ( ANLZ_ERR_UNEXPECTED );
            }

            RcvSize = DataSize - DataIndex; // Obtain remain bytes

            DecodeFuncState = pField->pDecode->pDecodeFunc(
                        DecodeFuncState,
```

```
                        &pData[DataIndex],
                        &RcvSize,
                        pField->DecodedBuf,
                        &DecodeFuncIndex,
                        pField->MaxDecodedBufSize,
                        pField->pDecode->StaticStorage);

            // pDataIndex is allways increamented:
            *pDataIndex = DataIndex + RcvSize;

            switch ( DecodeFuncState )
            {
              case DEC_STATE_COMPLETE:
                DecodeFuncState = DEC_STATE_BEGIN; // Prepare for next time.
                // further decoding/convert is performed on the decoded buffer itself:
                DecodeAndConvertMultField ( pField, DecodeFuncIndex, pField->DecodedBuf );
                *pMultiples = DecodeFuncIndex;
                DecodeFuncIndex = 0;
                break; // Ok decoding complete !

              case DEC_STATE_ERR_SYNC:
                DecodeFuncState = DEC_STATE_BEGIN; // Prepare for next time.
                DecodeFuncIndex = 0;
                return ( ANLZ_ERR_DFUNC_OUT_OF_SYNC );

              case DEC_STATE_ERR_FCS:
                DecodeFuncState = DEC_STATE_BEGIN; // Prepare for next time.
                DecodeFuncIndex = 0;
                return ( ANLZ_ERR_DFUNC_INVALID_FCS );

              case DEC_STATE_ERR_TOO_BIG:
                DecodeFuncState = DEC_STATE_BEGIN; // Prepare for next time.
                DecodeFuncIndex = 0;
                return ( ANLZ_ERR_DECODE_BUF_TOO_SMALL );

              default: return ( ANLZ_ERR_DFUNC_NOT_FINISHED );
            }

            break;

    default :  DEBUG_ERR ( " DecodeField: Unknown Field' Mult ! ");
            return ( ANLZ_ERR_UNEXPECTED );
  }

  return ( 0 ); // Ok, decoding success.
}

/***********************************************************************
 *
 * NAME:        TChannelProtocolAnalyzer::MatchMultField
 *
 * DESCRIPTION:   Match on all multiples of a field.
 *
 *           pField - new protocol layer.
 *           pInData - Node's non analyzed data bytes (yet not decoded) .
 *           Multiples - Multiples of field.
 *
 * RETURN VALUE:  Return TRUE for match or FALSE for no match.
 *
 * NOTES:        This routine does not handles a decoding function
 *           Size of InData always remains the same as OutData.
 *
 ***********************************************************************/

bool __fastcall TChannelProtocolAnalyzer::MatchMultField ( PT_PROTO_FIELD pField, DWORD Multiples, PT_SYNC_VALUE
pSyncVal )
{
  DWORD i;
  BYTE FieldSize, *pOutData;
```

```
if ( pSyncVal->Sync != SF_ALL ) // step in only if need
{
  FieldSize = pField->Type;
  pOutData = pField->DecodedBuf;

  for ( i = 0; i < Multiples; i++ )
  {
    // Event if there is only one that doesn't match -> return FALSE:
    if ( !MatchSingleField ( pField, pSyncVal, pOutData ) )
      return( FALSE );
    pOutData += FieldSize;
  }
}

return( TRUE ); // Ok all bytes matches
}

/****************************************************************************
*
* NAME:        TChannelProtocolAnalyzer::MatchBits
*
* DESCRIPTION:   Match bits partition node.
*
*          pField - new protocol layer.
*          pBits - Node' Specific bits info.
*          Multiples - Multiples of field.
*
* RETURN VALUE:  Return TRUE for match or FALSE for no match.
*
* NOTES:      This routine is applied only if pBits != NULL.
*
****************************************************************************/

bool __fastcall TChannelProtocolAnalyzer::MatchBits ( PT_PROTO_FIELD pField, PT_BITS pBits, PT_SYNC_VALUE
pSyncVal )
{
  BYTE *pOutData, *pInData, ShiftR, ShiftL;
  DWORD Triple;

  if ( pBits->pFirst == NULL )
    pInData = pField->DecodedBuf; // first partition:
  else
    pInData = pBits->pFirst->pField->DecodedBuf; // not the first partition:
  pOutData = pBits->DecodedBits;

  /*******************************************************************/
  /* prepare DecodedBits buffer so that it will contain specified bits */
  /* boundries:                                  */
  /*                                             */
  /* Ex: prepare bits index 5 to 9:              */
  /*                                             */
  /* bits...  0,1,2,3,4,5,6,7,8,9,10,11,12,13....  */
  /*                                             */
  /* InData:  X,X,X,X,X,1,1,1,0,1,X, X, X, X....   */
  /* OutData: 1,1,1,0,1,0,0,0,0,0,0, 0, 0, 0....   */
  /*                                             */
  /* operation: 1.shift left till end.            */
  /*        2.shift back right till start.        */
  /*******************************************************************/

  //ShiftR = 7 - ToBitIndex;
  //ShiftL = 7 - ( ToBitIndex - FromBitIndex );

  switch ( pField->Type )
  {
    case TF_BYTE:
      ShiftL = (BYTE)(7 - pBits->ToBitIndex);
      ShiftR = (BYTE)(7 - ( pBits->ToBitIndex - pBits->FromBitIndex ));
```

```
    *(BYTE *)pOutData = (BYTE)( (BYTE)( (*(BYTE *)pInData) << ShiftL ) >> ShiftR ) ;
    break;
  case TF_WORD:
    ShiftL = (BYTE)(15 - pBits->ToBitIndex);
    ShiftR = (BYTE)(15 - ( pBits->ToBitIndex - pBits->FromBitIndex ));
    *(WORD *)pOutData = (WORD)( (WORD)( (*(WORD *)pInData) << ShiftL ) >> ShiftR ) ;
    break;
  case TF_TRIPLE:
    Triple = pInData[0] + (pInData[1]<<8) + (pInData[2]<<16);
    ShiftL = (BYTE)(23 - pBits->ToBitIndex);
    ShiftR = (BYTE)(23 - ( pBits->ToBitIndex - pBits->FromBitIndex ));
    Triple = ( Triple << ShiftL ) & 0xffffff;
    Triple = ( Triple >> ShiftR );
    pOutData[2] = (BYTE)((Triple>>16) & 0xff); // most
    pOutData[1] = (BYTE)((Triple>>8) & 0xff); // middle
    pOutData[0] = (BYTE)((Triple) & 0xff);    // least
    break;
  case TF_DWORD:
    ShiftL = (BYTE)(31 - pBits->ToBitIndex);
    ShiftR = (BYTE)(31 - ( pBits->ToBitIndex - pBits->FromBitIndex ));
    *(DWORD *)pOutData = ( (DWORD)( (*(DWORD *)pInData) << ShiftL ) >> ShiftR ) ;
    break;
  case TF_LARGE:
    ShiftL = (BYTE)(63 - pBits->ToBitIndex);
    ShiftR = (BYTE)(63 - ( pBits->ToBitIndex - pBits->FromBitIndex ));
    *(UINT64 *)pOutData = (UINT64)( (UINT64)( (*(UINT64 *)pInData) << ShiftL ) >> ShiftR ) ;
    break;

  default :     DEBUG_ERR ( " MatchBits: Unknown Type ! "); return ( FALSE );
 }

 return ( MatchSingleField ( pField, pSyncVal, pOutData ) );
}


/**********************************************************************
 *
 * NAME:        TChannelProtocolAnalyzer::MatchSingleField
 *
 * DESCRIPTION:  Match for a single specific field.
 *
 *          pField - pointer to field.
 *          pSyncVal - pointer to synchronization record.
 *          pOutData - pointer to next position of converted and
 *                decoded buffer.
 *
 * RETURN VALUE:  Return TRUE for match or FALSE for no match.
 *
 * NOTES:       This routine does not handles a decoding function
 *          Size of InData always remains the same as OutData.
 *          Calling function must ensure ther's enough decoded
 *          buffer space.
 *
 **********************************************************************/

bool __fastcall TChannelProtocolAnalyzer::MatchSingleField ( PT_PROTO_FIELD pField, PT_SYNC_VALUE pSyncVal, BYTE
*pOutData )
{
  PT_SWITCH_VALUE pSV; // pointer to Switch Value structure
  DWORD Triple;

  /*****************************************************/
  /* Process:                    */
  /* -------                     */
  /* try to match according to synchronization info.   */
  /*****************************************************/

  switch ( pField->Type )
  {
    case TF_BYTE:  switch ( pSyncVal->Sync )
```

```
            {
              case SF_ALL    : return ( TRUE );
              case SF_FIXED  : if ( *(BYTE *)pOutData == *(BYTE *)pSyncVal->Fixed )
                                   return ( TRUE );
                               return ( FALSE ); // no match
              case SF_RANGE  : if ( ( *(BYTE *)pOutData >= *(BYTE *)pSyncVal->LLimit ) &&
                                   ( *(BYTE *)pOutData <= *(BYTE *)pSyncVal->HLimit ) )
                                   return ( TRUE );
                               return ( FALSE ); // no match
              case SF_MASK   : if ( *(BYTE *)pOutData & *(BYTE *)pSyncVal->Mask )
                                   return ( TRUE );
                               return ( FALSE ); // no match
              case SF_SWITCH : pSV = pSyncVal->pFirst;
                               while ( pSV != NULL )
                               {
                                 if ( *(BYTE *)pOutData == *(BYTE *)pSV->Fixed )
                                   return ( TRUE );
                                 pSV = pSV->pNext;
                               }
                               return ( FALSE ); // no match
              case SF_FCS    : DEBUG_ERR ( " MatchSingleField (TF_BYTE): SF_FCS not handled !"); return (FALSE); //
no match
              default        : DEBUG_ERR ( " MatchSingleField (TF_BYTE): Unknown Sync value !"); return (FALSE); // no
match
            }

    case TF_WORD:  switch ( pSyncVal->Sync )
            {
              case SF_ALL    : return ( TRUE );
              case SF_FIXED  : if ( *(WORD *)pOutData == *(WORD *)pSyncVal->Fixed )
                                   return ( TRUE );
                               return ( FALSE ); // no match
              case SF_RANGE  : if ( ( *(WORD *)pOutData >= *(WORD *)pSyncVal->LLimit ) &&
                                   ( *(WORD *)pOutData <= *(WORD *)pSyncVal->HLimit ) )
                                   return ( TRUE );
                               return ( FALSE ); // no match
              case SF_MASK   : if ( *(WORD *)pOutData & *(WORD *)pSyncVal->Mask )
                                   return ( TRUE );
                               return ( FALSE ); // no match
              case SF_SWITCH : pSV = pSyncVal->pFirst;
                               while ( pSV != NULL )
                               {
                                 if ( *(WORD *)pOutData == *(WORD *)pSV->Fixed )
                                   return ( TRUE );
                                 pSV = pSV->pNext;
                               }
                               return ( FALSE ); // no match
              case SF_FCS    : DEBUG_ERR ( " MatchSingleField (TF_WORD): SF_FCS not handled !"); return (FALSE); //
no match
              default        : DEBUG_ERR ( " MatchSingleField (TF_WORD): Unknown Sync value !"); return (FALSE); // no
match
            }

    case TF_TRIPLE: Triple = pOutData[0] + (pOutData[1]<<8) + (pOutData[2]<<16);
            // Notice that 4'th byte of each pSyncVal' field must be
            // initialized with zero else DWORD conversion is invalid.
            switch ( pSyncVal->Sync )
            {
              case SF_ALL    : return ( TRUE );
              case SF_FIXED  : if ( Triple == *(DWORD *)pSyncVal->Fixed )
                                   return ( TRUE );
                               return ( FALSE ); // no match
              case SF_RANGE  : if ( ( Triple >= *(DWORD *)pSyncVal->LLimit ) &&
                                   ( Triple <= *(DWORD *)pSyncVal->HLimit ) )
                                   return ( TRUE );
                               return ( FALSE ); // no match
              case SF_MASK   : if ( Triple & *(DWORD *)pSyncVal->Mask )
                                   return ( TRUE );
                               return ( FALSE ); // no match
              case SF_SWITCH : pSV = pSyncVal->pFirst;
```

```
                    while ( pSV != NULL )
                    {
                      if ( Triple == *(DWORD *)pSV->Fixed )
                        return ( TRUE );
                      pSV = pSV->pNext;
                    }
                    return ( FALSE ); // no match
            case SF_FCS    : DEBUG_ERR ( " MatchSingleField (TF_TRIPLE): SF_FCS not handled !"); return (FALSE); //
no match
            default        : DEBUG_ERR ( " MatchSingleField (TF_TRIPLE): Unknown Sync value !"); return (FALSE); // no
match
          }

    case TF_DWORD:  switch ( pSyncVal->Sync )
          {
            case SF_ALL    : return ( TRUE );
            case SF_FIXED  : if ( *(DWORD *)pOutData == *(DWORD *)pSyncVal->Fixed )
                        return ( TRUE );
                    return ( FALSE ); // no match
            case SF_RANGE  : if ( ( *(DWORD *)pOutData >= *(DWORD *)pSyncVal->LLimit ) &&
                        ( *(DWORD *)pOutData <= *(DWORD *)pSyncVal->HLimit ) )
                        return ( TRUE );
                    return ( FALSE ); // no match
            case SF_MASK   : if ( *(DWORD *)pOutData & *(DWORD *)pSyncVal->Mask )
                        return ( TRUE );
                    return ( FALSE ); // no match
            case SF_SWITCH : pSV = pSyncVal->pFirst;
                    while ( pSV != NULL )
                    {
                      if ( *(DWORD *)pOutData == *(DWORD *)pSV->Fixed )
                        return ( TRUE );
                      pSV = pSV->pNext;
                    }
                    return ( FALSE ); // no match
            case SF_FCS    : DEBUG_ERR ( " MatchSingleField (TF_DWORD): SF_FCS not handled !"); return (FALSE);
// no match
            default        : DEBUG_ERR ( " MatchSingleField (TF_DWORD): Unknown Sync value !"); return (FALSE); //
no match
          }

    case TF_LARGE:  switch ( pSyncVal->Sync )
          {
            case SF_ALL    : return ( TRUE );
            case SF_FIXED  : if ( *(UINT64 *)pOutData == *(UINT64 *)pSyncVal->Fixed )
                        return ( TRUE );
                    return ( FALSE ); // no match
            case SF_RANGE  : if ( ( *(UINT64 *)pOutData >= *(UINT64 *)pSyncVal->LLimit ) &&
                        ( *(UINT64 *)pOutData <= *(UINT64 *)pSyncVal->HLimit ) )
                        return ( TRUE );
                    return ( FALSE ); // no match
            case SF_MASK   : if ( *(UINT64 *)pOutData & *(UINT64 *)pSyncVal->Mask )
                        return ( TRUE );
                    return ( FALSE ); // no match
            case SF_SWITCH : pSV = pSyncVal->pFirst;
                    while ( pSV != NULL )
                    {
                      if ( *(UINT64 *)pOutData == *(UINT64 *)pSV->Fixed )
                        return ( TRUE );
                      pSV = pSV->pNext;
                    }
                    return ( FALSE ); // no match
            case SF_FCS    : DEBUG_ERR ( " MatchSingleField (TF_LARGE): SF_FCS not handled !"); return (FALSE); //
no match
            default        : DEBUG_ERR ( " MatchSingleField (TF_LARGE): Unknown Sync value !"); return (FALSE); // no
match
          }

    default :   DEBUG_ERR ( " MatchSingleField: Unknown Type ! "); return ( FALSE );
  }
}
```

```
/**************************************************************************
 *
 * NAME:        TChannelProtocolAnalyzer::DecodeAndConvertMultField
 *
 * DESCRIPTION:   Decode and convert all multiples of field.
 *                Also verifys that DecodedBuf is big enough to
 *                contain decoded data.
 *
 *                pField - field to convert.
 *                Multiples - Multiples of field.
 *                pInData - Node's non analyzed data bytes (yet not decoded) .
 *
 * RETURN VALUE:  TRUE for ok, or FALSE for field's DecodedBuf too small.
 *
 * NOTES:         This routine does not handles a decoding function
 *                Size of InData always remains the same as OutData.
 *
 **************************************************************************/

bool __fastcall TChannelProtocolAnalyzer::DecodeAndConvertMultField ( PT_PROTO_FIELD pField, DWORD Multiples, BYTE
*pInData )
{
  DWORD i;
  BYTE FieldSize;
  BYTE *pOutData;

  FieldSize = pField->Type;
  pOutData = pField->DecodedBuf;

  if ( Multiples * FieldSize > pField->MaxDecodedBufSize )
    return( FALSE );

  // Scan all data (according to Multiples):
  for ( i = 0; i < Multiples; i++ )
  {
    DecodeAndConvertSingleField ( pField, pInData, pOutData );
    pInData += FieldSize;
    pOutData += FieldSize;
  }

  return( TRUE );
}


/**************************************************************************
 *
 * NAME:        TChannelProtocolAnalyzer::DecodeAndConvertSingleField
 *
 * DESCRIPTION:   Decode and Convert Single multiples of a specific field.
 *
 *                pField - field to convert.
 *                pInData - Node's non analyzed data bytes (yet not decoded) .
 *                pOutData - Node's analyzed and decoded data bytes.
 *
 * RETURN VALUE:  None.
 *
 * NOTES:         This routine does not handles a decoding function
 *                Size of InData always remains the same as OutData.
 *                Calling function must ensure ther's enough decoded
 *                buffer space.
 *
 **************************************************************************/

void __fastcall TChannelProtocolAnalyzer::DecodeAndConvertSingleField ( PT_PROTO_FIELD pField, BYTE *pInData, BYTE
*pOutData )
{
  /*****************************************************/
  /* Process:                              */
```

```
/* ------- */
/* 1. Copy to out buffer according to bits' order: */
/* 2. Perform internal decoding operation (if BYTE).*/
/**************************************************/

switch ( pField->Type )
{
  case TF_BYTE:  pOutData[0] = pInData[0];

            switch ( pField->Convert )  // Special convertions for BYTE Type:
            {
              case CON_NO_CONVERT :  break;
              case CON_ADD       :  pOutData[0] += pField->ConvertVal; break;
              case CON_SUB       :  pOutData[0] -= pField->ConvertVal; break;
              case CON_NOT_LOGIC  :  pOutData[0] = (BYTE)~(pOutData[0]); break;
              case CON_OR_LOGIC   :  pOutData[0] |= pField->ConvertVal; break;
              case CON_AND_LOGIC  :  pOutData[0] &= pField->ConvertVal; break;
              case CON_XOR_LOGIC  :  pOutData[0] ^= pField->ConvertVal; break;
              case CON_NAND_LOGIC :  pOutData[0] = (BYTE)(~(pOutData[0] & pField->ConvertVal)); break;
              default           :  DEBUG_ERR ( " DecodeAndConvertSingleField: Unknown Convert type !"); break;
            }
            break;

  case TF_WORD:  if ( pField->BitDir == BD_MSB_FIRST )
            {  // Motorola like (Big indian) --> swap
              pOutData[0] = pInData[1];
              pOutData[1] = pInData[0];
            }
            else
              *(WORD *)pOutData = *(WORD *)pInData;

            //switch ( pField->Convert )  // Special convertions for WORD Type:
            //{
            //  case CON_NO_CONVERT :  break;
            //  case CON_ADD       :  *(WORD *)pOutData += *(WORD *)pField->ConvertVal; break;
            //  case CON_SUB       :  *(WORD *)pOutData -= *(WORD *)pField->ConvertVal; break;
            //  case CON_NOT_LOGIC  :  *(WORD *)pOutData[0] = ~*(WORD *)pOutData; break;
            //  case CON_OR_LOGIC   :  *(WORD *)pOutData |= *(WORD *)pField->ConvertVal; break;
            //  case CON_AND_LOGIC  :  *(WORD *)pOutData &= *(WORD *)pField->ConvertVal; break;
            //  case CON_XOR_LOGIC  :  *(WORD *)pOutData ^= *(WORD *)pField->ConvertVal; break;
            //  case CON_NAND_LOGIC :  *(WORD *)pOutData = ~(*(WORD *)pOutData & *(WORD
*)pField->ConvertVal); break;
            //  default:          :  DEBUG_ERR ( " DecodeAndConvertSingleField: Unknown Convert type !"); break;
            //
            //}

            break;

  case TF_TRIPLE: if ( pField->BitDir == BD_MSB_FIRST )
            {  // Motorola like (Big indian) --> swap
              pOutData[0] = pInData[2];
              pOutData[1] = pInData[1];
              pOutData[2] = pInData[0];
            }
            else
            {
              pOutData[0] = pInData[0];
              pOutData[1] = pInData[1];
              pOutData[2] = pInData[2];
            }
            break;

  case TF_DWORD:  if ( pField->BitDir == BD_MSB_FIRST )
            {  // Motorola like (Big indian) --> swap
              pOutData[0] = pInData[3];
              pOutData[1] = pInData[2];
              pOutData[2] = pInData[1];
              pOutData[3] = pInData[0];
            }
            else
```

```
            *((DWORD *)pOutData) = *((DWORD *)pInData);
          break;

  case TF_LARGE:  if ( pField->BitDir == BD_MSB_FIRST )
          { // Motorola like (Big indian) --> swap
            pOutData[0] = pInData[7];
            pOutData[1] = pInData[6];
            pOutData[2] = pInData[5];
            pOutData[3] = pInData[4];
            pOutData[4] = pInData[3];
            pOutData[5] = pInData[2];
            pOutData[6] = pInData[1];
            pOutData[7] = pInData[0];
          }
          else
            *((UINT64 *)pOutData) = *((UINT64 *)pInData);
          break;

   default :    DEBUG_ERR ( " DecodeAndConvertSingleField: Unknown Type ! ");

  }
}


/*********************************************************************
 *
 * NAME:        TChannelProtocolAnalyzer::CheckNodeConditions
 *
 * DESCRIPTION:   Checks all conditions in a given node.
 *
 * RETURN VALUE:  None.
 *
 * NOTES:       Called upon analysis synchronized on pNode.
 *
 *********************************************************************/

void __fastcall TChannelProtocolAnalyzer::CheckNodeConditions( PT_PROTO_NODE pNode, DWORD Multiples )
{
  register PT_HOOKED_CONDITIONS pHookedCondition = pNode->pFirstHookedCondition;

  // Scan all conditions...
  while ( pHookedCondition != NULL )
  {
    pHookedCondition->pOnCheckConditionFunc( pNode, pHookedCondition->pConditionNode, Multiples );
    pHookedCondition = pHookedCondition->pNext;
  }
}


/*********************************************************************
 *
 * NAME:        TChannelProtocolAnalyzer::CheckProtocolConditions
 *
 * DESCRIPTION:   Checks all conditions in a given protocol.
 *
 * RETURN VALUE:  None.
 *
 * NOTES:       Called upon analysis synchronized on pProtocol.
 *
 *********************************************************************/

void __fastcall TChannelProtocolAnalyzer::CheckProtocolConditions( PT_PROTO_TREE pProtocol )
{
  register PT_HOOKED_CONDITIONS pHookedCondition = pProtocol->pFirstHookedCondition;

  // Scan all conditions...
  while ( pHookedCondition != NULL )
  {
    pHookedCondition->pOnCheckConditionFunc( pProtocol, pHookedCondition->pConditionNode, 0/*Multiples are dummy*/);
    pHookedCondition = pHookedCondition->pNext;
        }
```

```
  }
}

/****************************************************************************
 *
 * NAME:        TChannelProtocolAnalyzer::NodeValueAsDWORD
 *
 * DESCRIPTION:  Retuens node's first decoded value converted to DWORD.
 *
 *          pNode - protocol node from which to peek decoded value.
 *
 * RETURN VALUE:  None.
 *
 * NOTES:      None.
 *
 ****************************************************************************/

DWORD __fastcall TChannelProtocolAnalyzer::NodeValueAsDWORD ( PT_PROTO_NODE pNode )
{
  BYTE *Buf;

  if ( pNode->pBits )
    Buf = pNode->pBits->DecodedBits;
  else
    Buf = pNode->pField->DecodedBuf;

  switch ( pNode->pField->Type )
  {
    case TF_BYTE:  return ( (DWORD)(*((BYTE *)Buf)) );
    case TF_WORD:  return ( (DWORD)(*((WORD *)Buf)) );
    case TF_TRIPLE: return (  (DWORD)Buf[0] +
                     (((DWORD)Buf[1])<<8) +
                     (((DWORD)Buf[2])<<16));
    case TF_DWORD:  return ( *((DWORD *)Buf) );
    case TF_LARGE:  return ( (DWORD)(*((UINT64 *)Buf)) );
    default:     DEBUG_ERR ( " NodeValueAsDWORD: Unknown Type ! "); return (0);
  }
}


//---------------------------------------------------------------------------
#pragma package(smart_init)
```

INTERNATIONAL SEARCH REPORT

International application No.
PCT/IL00/00639

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : H02H 3/05
US CL : 714/39
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 714/1, 25, 37, 39

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EAST
search terms: digital, test, testing, protocol, user, select

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X -- Y	US 5,127,009 A (SWANSON) 30 June 1992, Figs 1a, 1b and 2 and col. 11 line 42 - col. 19 line 55.	1 -- 2-5, 26
Y	US 5,027,343 A (CHAN et al) 25 June 1991, Fig. 5 and col. 3 - col. 12	2-5, 26

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search
07 FEBRUARY 2001

Date of mailing of the international search report
05 MAR 2001

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
ROBERT W. BEAUSOLIEL JR.
Telephone No. (703) 308-7090

Form PCT/ISA/210 (second sheet) (July 1998)★

INTERNATIONAL SEARCH REPORT

International application No.
PCT/IL00/00639

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This international report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☒ Claims Nos.: 30
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

the specific invention claimed by claim 30 is unclear. See PCT rule 5.

3. ☒ Claims Nos.: 6-25, 27-29
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest
☐ The additional search fees were accompanied by the applicant's protest.
☐ No protest accompanied the payment of additional search fees.